МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ІНСТРУМЕНТУ ДЛЯ ІНТЕРАКТИВНОЇ ВІЗУАЛІЗАЦІЇ ВЕКТОРНОЇ АЛГЕБРИ ЗАСОБАМИ TKINTER (PYTHON)»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
К. С. Бейз
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бейзу Кирилу Сергійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка навчального програмного інструменту для інтерактивної візуалізації векторної алгебри засобами Tkinter (Python)

керівник роботи Борю Сергій Юрійович, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування навчального програмного інструменту.
3. Розробка та тестування програмного інструменту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	17.02.2026	
2.	Збір вихідних даних.	20.02.2026	
3.	Обробка методичних та теоретичних	23.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	13.03.2026	
5.	Розробка третього розділу.	24.05.2026	
6.	Оформлення та нормоконтроль	05.06.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2026	

Студент _____ К. С. Бейз
(підпис) (ініціали та прізвище)

Керівник роботи _____ С. Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка навчального програмного інструменту для інтерактивної візуалізації векторної алгебри засобами Tkinter (Python)»: 123 с., 29 рис., 7 табл., 28 джерел, 4 додатків.

ВЕКТОРНА АЛГЕБРА, ІНТЕРАКТИВНА ВІЗУАЛІЗАЦІЯ, МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ, НАВЧАЛЬНИЙ ПРОГРАМНИЙ ІНСТРУМЕНТ, ОСВІТНІ ТЕХНОЛОГІЇ, PYTHON, TKINTER, 2D/3D ГРАФІКА, ANIMATION, EDUCATIONAL SOFTWARE

Об’єктом дослідження є процеси навчання векторної алгебри з використанням інтерактивних програмних засобів.

Предметом дослідження є методи та засоби створення навчального програмного інструменту для візуалізації векторних операцій у 2D та 3D просторах.

Метою роботи є розробка інструменту для інтерактивної візуалізації векторної алгебри, що забезпечує наочність математичних операцій і підвищує ефективність навчання.

Методи дослідження – аналіз наукової літератури, порівняльний аналіз існуючих програмних засобів, методи об’єктно-орієнтованого програмування, методи комп’ютерної графіки, алгоритми афінних та проекційних перетворень, модульне та інтеграційне тестування програмного забезпечення.

У роботі проаналізовано роль векторної алгебри та існуючі засоби візуалізації, сформульовано вимоги до системи. Спроєктовано архітектуру інструменту, розроблено алгоритми відображення векторів і операцій. Реалізовано програмний засіб на Python і Tkinter з підтримкою 2D/3D візуалізації та анімації. Проведене тестування підтвердило коректність роботи та зручність використання системи.

SUMMARY

Bachelor's Qualifying Thesis «Development of an Educational Software Tool for Interactive Visualization of Vector Algebra Using Tkinter (Python)»: 123 pages, 29 figures, 7 tables, 28 sources, 4 supplements.

VECTOR ALGEBRA, INTERACTIVE VISUALIZATION, MATHEMATICAL MODELING, EDUCATIONAL SOFTWARE TOOL, EDUCATIONAL TECHNOLOGIES, PYTHON, TKINTER, 2D/3D GRAPHICS, ANIMATION, EDUCATIONAL SOFTWARE

The object of the research is the learning process of vector algebra using interactive software tools.

The subject of the research is methods and tools for developing an educational software application for visualizing vector operations in 2D and 3D spaces.

The aim of the work is to develop a tool for interactive visualization of vector algebra that provides a clear representation of mathematical operations and improves learning effectiveness.

Research methods include literature analysis, comparative analysis of existing software tools, object-oriented programming methods, computer graphics methods, affine and projection transformation algorithms, and software unit and integration testing.

The work analyzes the role of vector algebra and existing visualization tools, and defines system requirements. The system architecture is designed, and algorithms for vector and operation visualization are developed. The software is implemented in Python and Tkinter with support for 2D/3D visualization and animation. Testing confirmed the correctness of the system and its usability.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області та постановка задачі 9
1.1 Векторна алгебра як розділ математичного аналізу та її навчальна роль……..... 9
1.2 Програмні засоби для візуалізації математичних об’єктів..... 12
1.3 Аналіз існуючих інтерактивних навчальних рішень..... 14
1.4 Постановка задачі 19
1.5 Висновки 21
2 Проєктування навчального програмного інструменту 22
2.1 Розробка функціональних вимог до програми 22
2.2 Структурне проєктування системи 28
2.3 Розробка алгоритмів візуалізації векторів та операцій над ними..... 38
2.4 Висновки 57
3 Розробка та тестування програмного інструменту..... 59
3.1 Обґрунтування вибору технологій розробки (Python, Tkinter, Matplotlib)..... 59
3.2 Реалізація інтерфейсу користувача засобами Tkinter..... 61
3.3 Реалізація інтерактивної візуалізації векторних операцій..... 67
3.4 Тестування функціональності та оцінка зручності використання..... 73
3.5 Висновки 74
Висновки 75
Перелік посилань..... 77
Додаток А Лістинг реалізації інтерфейсу користувача засобами Tkinter 80
Додаток Б Лістинг реалізація інтерактивної візуалізації векторних операцій90
Додаток В Інтерфейс програми 109
Додаток Г Лістинг тестування програми..... 113

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується активною інтеграцією програмних засобів у освітній процес, зокрема у викладання математичних дисциплін. Однією з ключових проблем традиційного навчання вищої математики є недостатній рівень наочності та інтерактивності при вивченні абстрактних понять. Особливо це стосується розділів, пов’язаних із векторною алгеброю, де студенти часто стикаються зі складнощами у просторовому уявленні векторів, виконанні операцій над ними та інтерпретації геометричного змісту математичних формул.

У зв’язку з цим актуальним є розроблення навчальних програмних інструментів, які забезпечують інтерактивну візуалізацію математичних об’єктів і дозволяють поєднувати теоретичні знання з їх практичною інтерпретацією. Використання таких засобів сприяє підвищенню якості засвоєння матеріалу, розвитку просторового мислення, а також підвищенню мотивації здобувачів освіти до навчання. Одним із перспективних напрямів є створення настільних навчальних застосунків із використанням мови програмування Python та бібліотеки Tkinter [1], що забезпечує доступність, кросплатформеність та простоту інтеграції з іншими інструментами візуалізації.

Актуальність теми обумовлена зростаючою потребою в цифрових освітніх ресурсах, які забезпечують інтерактивну підтримку навчального процесу з математичних дисциплін, зокрема векторної алгебри. Незважаючи на наявність певної кількості онлайн-симуляторів та графічних калькуляторів, більшість із них або є надто складними для навчального використання, або не забезпечують достатнього рівня інтерактивності та адаптації до навчальних програм закладів вищої освіти. Це створює необхідність у розробці спеціалізованого програмного інструменту, орієнтованого саме на освітні потреби.

Метою роботи є розробка навчального програмного інструменту для інтерактивної візуалізації операцій векторної алгебри із використанням мови

програмування Python та бібліотеки Tkinter, що дозволяє підвищити ефективність засвоєння навчального матеріалу.

Об’єктом дослідження є процес навчання векторної алгебри у закладах вищої освіти із використанням інформаційних технологій.

Предметом дослідження є методи та програмні засоби інтерактивної візуалізації векторних операцій у навчальному процесі.

Історичний розвиток даної проблематики пов’язаний із еволюцією комп’ютерної графіки та освітніх технологій. Починаючи від перших систем комп’ютерної алгебри та графічних пакетів, таких як MATLAB і Mathematica [2], до сучасних інтерактивних веб- та десктопних застосунків, спостерігається тенденція до підвищення доступності та інтерактивності навчальних інструментів. Особливу роль у цьому процесі відіграє Python як універсальна мова програмування [3], що активно використовується в освіті та наукових дослідженнях, а також бібліотеки Tkinter і Matplotlib, які дозволяють швидко створювати графічні інтерфейси та здійснювати наукову візуалізацію даних.

Наукова новизна роботи полягає у розробці навчального програмного інструменту, який забезпечує інтерактивну візуалізацію операцій векторної алгебри в реальному часі та підвищує наочність навчального матеріалу.

Практичне значення полягає у можливості використання розробленого застосунку під час вивчення математичних дисциплін у закладах вищої освіти, зокрема на лекційних, практичних заняттях і в самостійній роботі студентів.

Результати роботи можуть бути використані для вдосконалення методики викладання вищої математики та подальшого розвитку інтерактивних освітніх програмних засобів.

Отже, розроблений програмний інструмент спрямований на підвищення ефективності навчання та покращення розуміння основ векторної алгебри завдяки використанню сучасних засобів візуалізації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Векторна алгебра як розділ математичного аналізу та її навчальна роль

Векторна алгебра є одним із фундаментальних розділів вищої математики, який займає проміжне положення між елементарною алгеброю, аналітичною геометрією та математичним аналізом. Її ключова особливість полягає у вивченні математичних об’єктів – векторів, які характеризуються одночасно величиною та напрямом. Саме це відрізняє їх від скалярних величин і дозволяє використовувати векторний апарат для опису багатовимірних просторових процесів. Векторна алгебра формує базу для подальшого вивчення більш складних математичних структур, зокрема векторного аналізу, диференціальних рівнянь та математичного моделювання [4].

У структурі математичної освіти векторна алгебра виконує роль перехідного етапу від класичних обчислювальних методів до аналітичного опису просторових об’єктів. Вона забезпечує формування у студентів базових уявлень про координатні системи, просторові вектори та операції над ними, що є необхідним для подальшого засвоєння аналітичної геометрії та математичного аналізу.

До основних операцій векторної алгебри належать додавання і віднімання векторів, множення вектора на число, скалярний, векторний та мішаний добутки. Крім того, важливими є поняття довжини (модуля) вектора, напрямних косинусів, проєкцій та координатного подання векторів. Ці операції дозволяють формалізувати просторові залежності та описувати геометричні об’єкти алгебраїчними засобами [5].

Отже, векторна алгебра виступає своєрідним «містком» між геометричним інтуїтивним сприйняттям простору та строгим аналітичним математичним апаратом. Вона дозволяє перейти від візуального мислення до формального

математичного моделювання, що є ключовим етапом у підготовці здобувачів вищої освіти.

З теоретичного погляду векторна алгебра є основою для побудови багатьох розділів сучасної математики. Вона закладає фундаментальні поняття лінійності, простору та взаємного розташування об’єктів у багатовимірних системах координат. Саме на основі векторної алгебри формуються такі поняття математичного аналізу, як градієнт, дивергенція, ротор, а також елементи векторних і скалярних полів [6].

Крім того, векторна алгебра дозволяє перейти від геометричного опису явищ до їх строгого аналітичного представлення. Це особливо важливо для розуміння сучасних математичних моделей, де просторові процеси описуються за допомогою векторних функцій та операторів. Таким чином, вона є базою для розвитку математичного мислення, орієнтованого на моделювання складних систем [7].

У навчальному контексті це означає, що студенти не лише засвоюють окремі правила обчислень, а й формують системне розуміння математичної структури простору, що є важливим для подальшого вивчення математичного аналізу та прикладних дисциплін [5].

Векторна алгебра має широке прикладне застосування у природничих і технічних науках. Вона використовується у фізиці для опису сил, швидкостей, прискорень, електромагнітних полів, а також у механіці для моделювання руху тіл. У інженерії векторні методи застосовуються для аналізу конструкцій, розрахунку навантажень та моделювання складних систем.

Завдяки цьому векторна алгебра є невід’ємною складовою професійної підготовки студентів технічних спеціальностей. Вона формує здатність до математичного моделювання реальних процесів, що є ключовою компетентністю сучасного фахівця [4].

Крім того, у комп’ютерних науках векторна алгебра використовується у графіці, машинному навчанні, робототехніці та симуляційних системах. Векторні обчислення дозволяють описувати положення об’єктів у просторі, їх

рух та взаємодію, що робить цей математичний апарат надзвичайно актуальним у сучасних інформаційних технологіях.

У педагогічному аспекті векторна алгебра виконує низку важливих функцій. По-перше, вона сприяє розвитку просторового мислення, оскільки студенти працюють з геометричними образами, які необхідно уявляти в дво- та тривимірному просторі. По-друге, вона формує логічне мислення, адже розв’язання задач потребує послідовного застосування математичних правил і властивостей. По-третє, вона розвиває аналітичні навички, оскільки більшість задач вимагає поєднання геометричної інтерпретації з алгебраїчними обчисленнями [5; 6].

Важливою є також роль векторної алгебри у формуванні міжпредметних зв’язків. Зокрема, вона активно використовується у фізиці, інформатиці, технічній механіці та інших дисциплінах. Це дозволяє студентам краще розуміти практичну цінність математичних знань та їх застосування у реальних професійних ситуаціях [8].

Окремо слід відзначити, що векторна алгебра сприяє розвитку навичок абстрактного мислення. Перехід від конкретних геометричних образів до їх алгебраїчного опису формує у студентів здатність працювати з формалізованими моделями, що є необхідним для подальшого навчання у сфері математичного аналізу та прикладної математики [6].

Отже, векторна алгебра є базовим і водночас ключовим розділом математичної підготовки, який забезпечує зв’язок між геометричним та аналітичним описом просторових об’єктів. Вона має важливе теоретичне значення як основа для подальшого розвитку математичного аналізу, а також значну навчальну та прикладну цінність. Її вивчення сприяє формуванню у студентів просторового, логічного та аналітичного мислення, що є необхідним для успішного засвоєння подальших математичних і прикладних дисциплін.

1.2 Програмні засоби для візуалізації математичних об’єктів

Сучасна математична освіта дедалі більше спирається на спеціалізоване програмне забезпечення, здатне перетворювати абстрактні поняття у наочні інтерактивні форми. Згідно з визначенням, прийнятим в освітній науці, візуалізація – це демонстрація явища чи процесу у зручній для зорового сприйняття формі, а в контексті математики вона передбачає графічне, динамічне або просторове представлення об’єктів, функцій і структур [9]. Програмні засоби для реалізації такої візуалізації умовно поділяються на три основні категорії: системи комп’ютерної математики (СКМ), системи динамічної геометрії та веборієнтовані інструменти.

Системи комп’ютерної математики (Computer Mathematical System) охоплюють такі продукти, як MATLAB, MathCad, Maple, Wolfram Mathematica, Maxima [10]. Кожна з них орієнтована на певну специфіку роботи з математичними об’єктами.

Wolfram Mathematica – одна з найпотужніших систем символьної математики. Вона підтримує аналітичні перетворення виразів, побудову двовимірних і тривимірних графіків, роботу з диференціальними рівняннями, лінійною алгеброю та статистикою: диференціювання, інтегрування, пошук коренів рівнянь та їх систем, обчислення сум і рядів [11]. Система орієнтована передусім на наукові дослідження та вищу освіту і вимагає платної ліцензії.

Maple є потужним пакетом для символьних і чисельних обчислень. Система надає зручне середовище для комп’ютерних експериментів та дозволяє створювати інтегровані середовища за участю інших систем і мов програмування. Maple вважається могутнім засобом для вивчення диференціальних рівнянь та задач математичного аналізу у технічних університетах, оскільки об’єднує знання з алгебри, аналітичної геометрії та математичного аналізу [12].

MATLAB з інструментом Simulink є провідним рішенням для інженерних і наукових обчислень. Для дослідження технічних процесів найбільш

ефективним є використання MATLAB з широким класом предметно-орієнтованих бібліотек (toolbox) та засобом візуального моделювання Simulink [10]. MATLAB орієнтований насамперед на чисельні обчислення та моделювання, а не на символьну алгебру.

MathCad відрізняється від попередніх систем: це, скоріше, редактор математичних текстів із широкими можливостями символьних обчислень. MathCad не має повноцінної мови програмування, а можливості символьних обчислень запозичені з пакета Maple [10]. Перевагою є інтуїтивно зрозумілий інтерфейс, максимально наближений до традиційного запису математичних виразів, що робить систему доступнішою для студентів, які не мають досвіду програмування.

Спільним недоліком СКМ є висока вартість ліцензій (за винятком Maxima та GNU Octave), необхідність встановлення програмного забезпечення та значний поріг входу для початківців.

Системи динамічної геометрії орієнтовані безпосередньо на навчальний процес і дозволяють будувати «живі» геометричні конструкції, що реагують на маніпуляції користувача. Серед таких засобів виділяють DG, Cabri II Plus, Geometer’s SketchPad та GeoGebra [13].

GeoGebra – вільно поширюване програмне середовище динамічної математики, що поєднує алгебру, геометрію, математичний аналіз і статистику. Можливості програми дозволяють ефективно використовувати її з різною метою: швидко створювати якісні зображення математичних об’єктів – графіки функцій, геометричні фігури, формули, діаграми – та зберігати їх для подальшої демонстрації чи використання у мультимедійних презентаціях [14]. GeoGebra підтримує як локальне встановлення, так і роботу у браузері.

Gran – українське програмне рішення для графічного аналізу, що існує у трьох версіях: Gran1 (аналіз функцій), Gran-2D (геометричні об’єкти на площині) та Gran-3D (просторові об’єкти) [13]. Перевагою є повністю україномовний інтерфейс, однак функціональність суттєво обмеженіша порівняно з міжнародними аналогами.

З розвитком хмарних технологій широкого поширення набули засоби, що не потребують встановлення і працюють безпосередньо у веббраузері.

Desmos – безкоштовний вебзастосунок компанії Desmos Studio, що пропонує графічний калькулятор (2D), тривимірний калькулятор, науковий калькулятор і геометричний інструмент. Тривимірний калькулятор Desmos дозволяє будувати поверхні, криві та точки у тривимірному просторі, виконувати векторні операції, інтерактивно обертати та масштабувати графіки [15]. Desmos активно використовується у шкільній та університетській практиці для навчання алгебри, тригонометрії та початків аналізу.

Wolfram Alpha функціонує як обчислювальна система знань, що відповідає на математичні запити та генерує графіки й покрокові рішення. На відміну від повноцінних СКМ, він орієнтований на швидкий пошук результату, а не на дослідницьку роботу та навчання.

Узагальнений порівняльний аналіз показує, що кожна категорія засобів має свої переваги та обмеження. СКМ потужні, але дорогі й складні у освоєнні; системи динамічної геометрії зручні у навчанні, але обмежені тематично; веборієнтовані інструменти доступні, однак не забезпечують повного охоплення тем вищої математики і не мають вбудованого структурованого навчального контенту. Жоден із розглянутих засобів не поєднує в собі потужної математичної візуалізації, структурованого навчального курсу, автоматизованого оцінювання та повноцінного україномовного інтерфейсу в єдиному доступному вебзастосунку.

1.3 Аналіз існуючих інтерактивних навчальних рішень

Розвиток дистанційної освіти сформував значний ринок інтерактивних навчальних платформ. Математика через абстрактність понять і вимоги до логічного мислення є однією з найскладніших дисциплін для дистанційного

засвоєння, тому саме у цій галузі з'явилась найбільша кількість спеціалізованих рішень [16].

Khan Academy (рис. 1.1) – відкрита освітня платформа, що поєднує навчальні відеоролики з інтерактивними вправами і стала взірцем у galузі цифрової освіти. Безпосередньо поруч із навчальними матеріалами розміщені інтерактивні тести, на основі яких платформа автоматично відстежує набуті навички студента, і ці дані доступні як студенту, так і викладачу [17]. Платформа містить понад 4000 навчальних відео та охоплює математику від початкової арифметики до початків математичного аналізу. Суттєвим обмеженням для вітчизняної аудиторії є недостатньо розвинена україномовна версія та відсутність контенту, адаптованого до навчальних програм українських закладів вищої освіти.

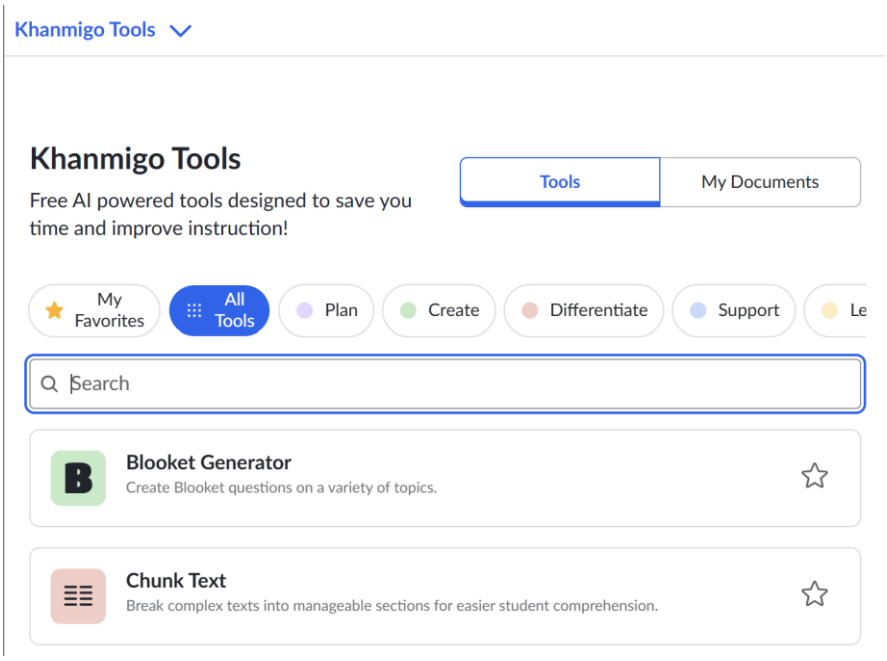


Рисунок 1.1 – Khan Academy

Coursera та edX – масові відкриті онлайн-платформи (MOOC). EdX (рис. 1.2) є спільним проектом Гарвардського університету та MTI і пропонує навчання за напрямами: комп'ютерні технології, математика, природничі науки та інші; успішним студентам видається сертифікат [18]. MOOC-платформи орієнтовані на проходження повних структурованих курсів від провідних

університетів, однак не забезпечують інтерактивної роботи з конкретними математичними об’єктами у реальному часі та позбавлені вбудованих засобів динамічної візуалізації.

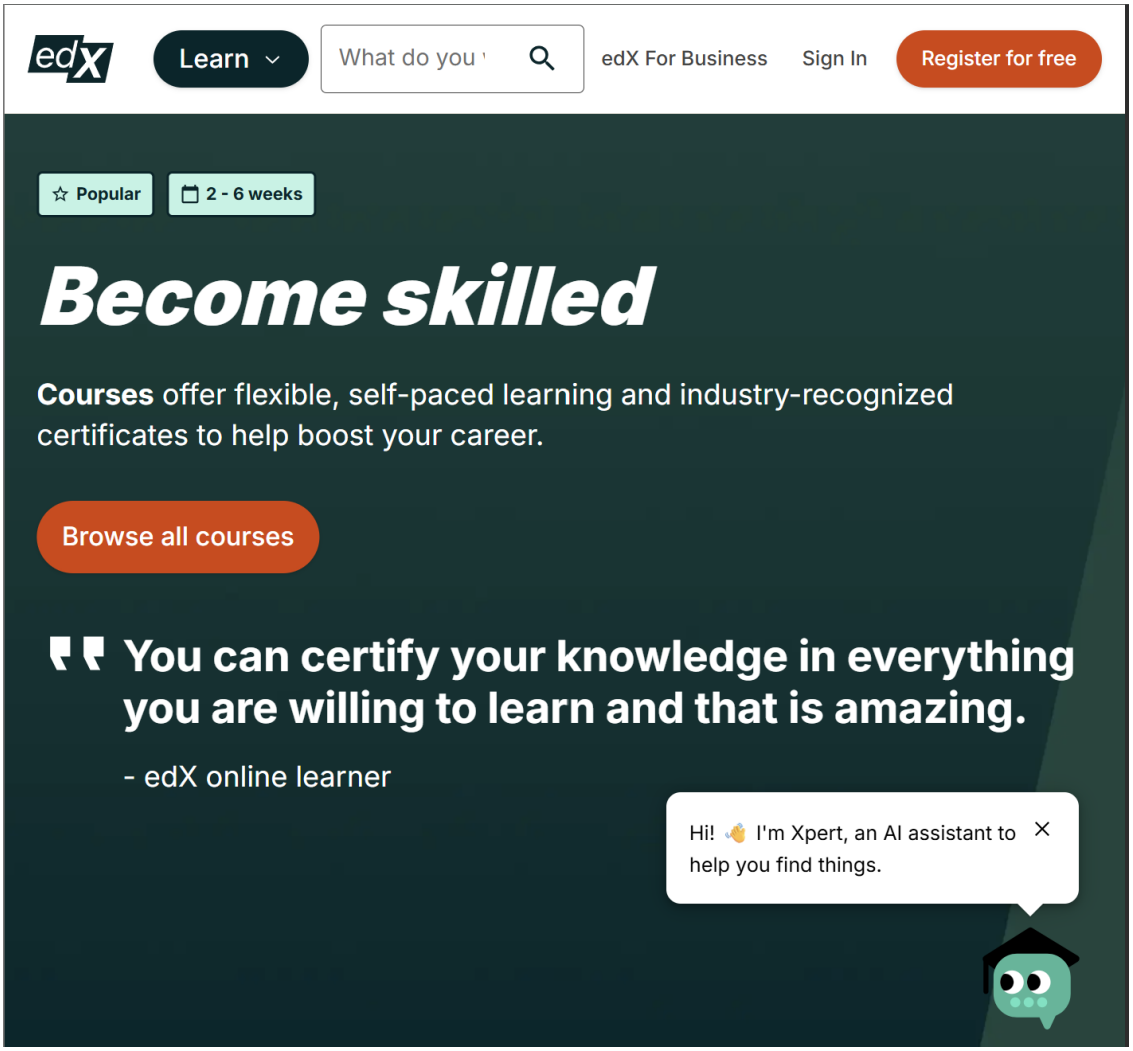


Рисунок 1.2 – Khan Academy

GeoGebra Classroom (рис. 1.3) – навчальний модуль платформи GeoGebra, що дозволяє викладачам створювати завдання з динамічними аплетами і стежити за прогресом студентів у реальному часі [19]. Перевагами є потужна динамічна візуалізація і можливість спільної роботи; недоліком – відсутність власного структурованого теоретичного курсу, покрокових пояснень і системи адаптивного оцінювання. GeoGebra Classroom є доповненням до існуючих навчальних матеріалів, а не самодостатньою навчальною системою.

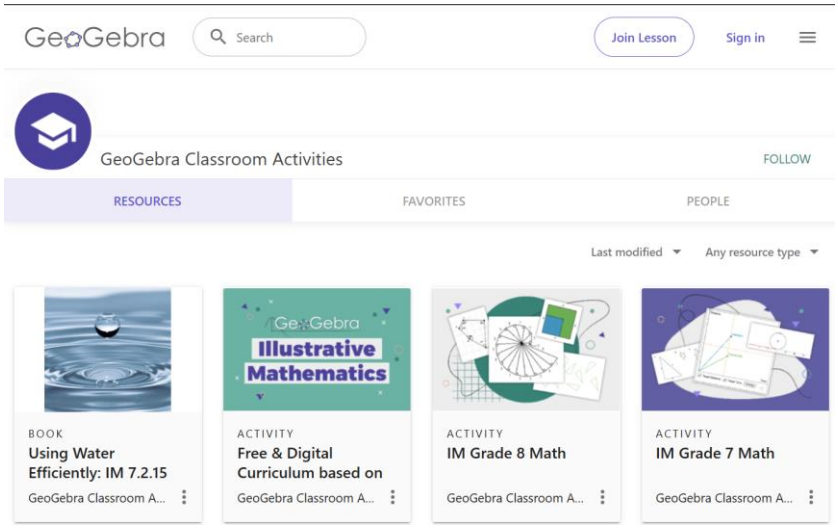


Рисунок 1.3 – GeoGebra Classroom

Desmos Activity Builder (рис. 1.4) надає вчителям інструментарій для розробки цифрових інтерактивних уроків із вбудованими графічними елементами [20]. Проте, як і GeoGebra Classroom, система вимагає від педагога значних зусиль для підготовки контенту, не має власних навчальних матеріалів і не підтримує автоматичного відстеження прогресу студентів у довгостроковій перспективі.

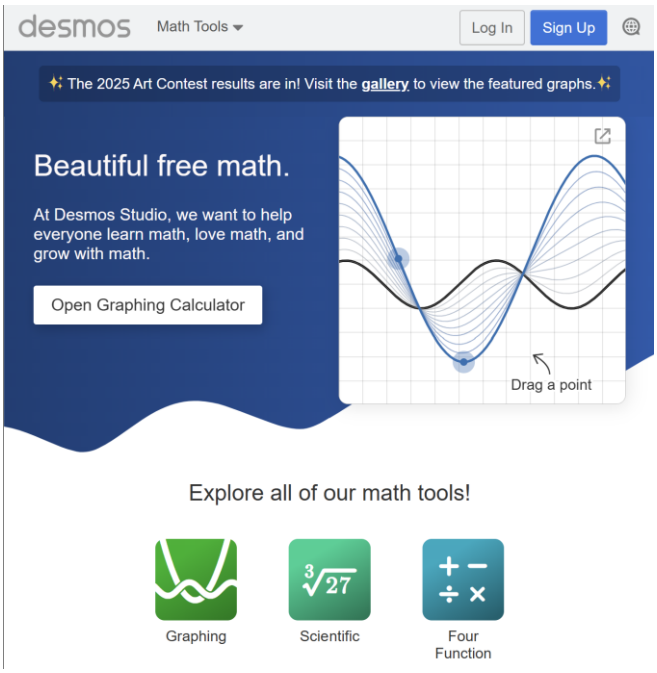


Рисунок 1.4 – Desmos Activity Builder

Wolfram Alpha Pro (рис. 1.5) пропонує покрокові рішення математичних задач і графіки, однак функціонує як обчислювальний довідник, а не як навчальна система: відсутні прогрес студента, зворотний зв’язок у навчальному сенсі та послідовна структура курсу [21].

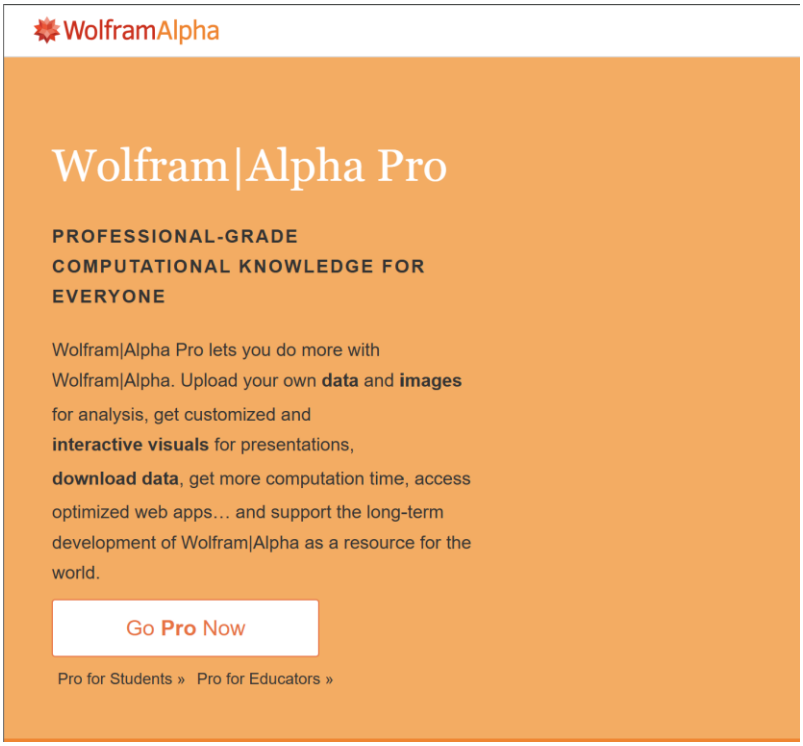


Рисунок 1.5 – Khan Academy

На підставі проведеного аналізу виявлено спільні недоліки розглянутих платформ, що обмежують їх застосування у контексті навчання вищої математики в Україні:

- відсутність повноцінного україномовного контенту з вищої математики, адаптованого до вітчизняних навчальних програм;
- розрив між засобами візуалізації та навчальним контентом – вони існують окремо, а не інтегровані в єдину систему;
- обмежена підтримка адаптивного навчання: відсутність індивідуалізованого темпу й рівня складності завдань;
- недостатній або надто загальний зворотний зв’язок: студент отримує лише «вірно/невірно» без пояснення помилки;

- необхідність встановлення програмного забезпечення (для СКМ) або залежність від комерційних ліцензій;
- відсутність ролевої моделі, яка б дозволяла викладачу керувати групами студентів, призначати завдання та переглядати аналітику успішності.

Тобто автоматизовані системи суттєво зменшують витрати часу викладачів, оскільки забезпечують оперативну перевірку результатів та надання зворотного зв’язку, усуваючи необхідність у ручній перевірці, завдяки чому викладачі можуть зосереджуватися на творчих аспектах освітнього процесу.

1.4 Постановка задачі

Метою розробки є створення навчального програмного інструменту для інтерактивної візуалізації векторної алгебри засобами Python із використанням бібліотеки Tkinter та інструментів графічної побудови, що дозволяє підвищити наочність навчального матеріалу та полегшити його засвоєння. Такий підхід має забезпечити поєднання графічного інтерфейсу з математичною моделлю, що оновлюється в режимі реального часу відповідно до дій користувача.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну область векторної алгебри та її роль у навчальному процесі;
- дослідити існуючі програмні засоби для візуалізації математичних об’єктів;
- визначити вимоги до функціональності та інтерфейсу навчального програмного інструменту;
- розробити структуру та архітектуру програмної системи;
- спроєктувати алгоритми візуалізації векторів та операцій над ними;
- реалізувати програмний інструмент із використанням Python, Tkinter та Matplotlib;

- провести тестування розробленого застосунку та оцінити його зручність і ефективність.

У ході аналізу теоретичних джерел було досліджено сучасний стан викладання векторної алгебри у закладах освіти характеризується наявністю певного розриву між теоретичним викладом матеріалу та його практичною й візуальною інтерпретацією. Хоча здобувачі освіти отримують достатній обсяг формальних знань щодо операцій над векторами, координатного подання та геометричних властивостей, на практиці часто виникають труднощі з уявленням просторових об’єктів і розумінням їхніх змін у динаміці. Це свідчить про необхідність використання більш наочних і інтерактивних засобів навчання, які дозволяють поєднати аналітичний підхід із візуальним представленням математичних об’єктів.

У зв’язку з цим виникає потреба у створенні спеціалізованого навчального програмного інструменту, який би забезпечував інтерактивну візуалізацію основних понять векторної алгебри та дозволяв користувачеві безпосередньо взаємодіяти з математичними об’єктами. Такий інструмент має не лише відображати вектори та операції над ними, а й забезпечувати зміну параметрів у реальному часі, що сприятиме кращому розумінню геометричної сутності математичних перетворень.

Під час формулювання задачі враховується необхідність поєднання простоти використання програмного засобу з його функціональною достатністю для навчальних цілей. Зокрема, система повинна бути орієнтована на студентів, які не мають глибоких знань у програмуванні, але потребують ефективного інструменту для вивчення векторної алгебри. Водночас вона має забезпечувати достатній рівень гнучкості для демонстрації базових і більш складних операцій над векторами у двовимірному та тривимірному просторах.

Отже, постановка задачі полягає у розробці інтерактивного навчального програмного середовища, яке поєднує математичну коректність, наочність подання інформації та зручність користування. Реалізація такого інструменту спрямована на підвищення ефективності засвоєння векторної алгебри та

формування у студентів більш глибокого просторового та аналітичного розуміння математичних об’єктів.

1.5 Висновки

У першому розділі було проведено аналіз предметної області дослідження та визначено основні передумови для розробки навчального програмного інструменту з інтерактивної візуалізації векторної алгебри. Розглянуто теоретичні засади векторної алгебри, її місце в системі математичних дисциплін та роль у формуванні просторового, логічного й аналітичного мислення студентів. Встановлено, що векторна алгебра є важливою складовою математичної підготовки та має широке застосування в технічних, природничих і комп’ютерних науках.

Проаналізовано сучасні засоби візуалізації математичних об’єктів, зокрема системи комп’ютерної математики, динамічної геометрії та веборієнтовані інструменти. Визначено їхні переваги й недоліки та встановлено, що існуючі рішення не завжди забезпечують одночасно наочність, зручність використання та орієнтацію на навчальні потреби студентів. Також було виконано аналіз існуючих інтерактивних навчальних платформ, що дозволило виявити основні проблеми сучасних освітніх рішень, серед яких недостатня інтеграція навчального контенту з інструментами візуалізації, обмежена підтримка україномовного середовища та відсутність спеціалізованих засобів для вивчення векторної алгебри. На основі проведеного аналізу сформовано функціональні та нефункціональні вимоги до майбутньої системи.

Результати дослідження стали підґрунтям для постановки задачі розробки навчального програмного інструменту засобами Python і Tkinter, призначеного для інтерактивної візуалізації векторів та операцій над ними. Сформульовані вимоги й визначені напрями реалізації є основою для подальшого проєктування архітектури програмного засобу, що розглядається в наступному розділі роботи.

2 ПРОЄКТУВАННЯ НАВЧАЛЬНОГО ПРОГРАМНОГО ІНСТРУМЕНТУ

2.1 Розробка функціональних вимог до програми

На етапі розробки функціональних вимог було сформовано загальну концепцію програмного засобу “ВекторВіз” як інтерактивного навчального середовища для візуалізації операцій над векторами. Основна мета цього етапу полягає у визначенні того, які саме функції повинна виконувати система, які користувачі з нею взаємодіятимуть та яким чином буде організована їхня робота.

Розроблюваний програмний інструмент “ВекторВіз” є desktop-застосунком, реалізованим мовою Python із використанням бібліотеки Tkinter для побудови графічного інтерфейсу користувача. Система орієнтована на навчальні цілі та забезпечує інтерактивну візуалізацію операцій над векторами у двовимірному та частково тривимірному просторі.

Основне призначення системи полягає у формуванні у користувачів наочного розуміння базових понять векторної алгебри. Зокрема, застосунок дозволяє виконувати математичні операції над векторами, спостерігати їх геометричну інтерпретацію та аналізувати отримані результати в режимі реального часу. Окремо варто зазначити, що система також може використовуватися як допоміжний інструмент під час лекційних і практичних занять або для самостійного навчання.

У процесі аналізу було визначено основні групи користувачів системи, які відрізняються за рівнем підготовки та цілями використання програмного продукту. Узагальнені характеристики подано в таблиці 2.1.

У таблиці наведено основні категорії користувачів системи, які визначають вимоги до її функціональності та способу використання.

Студенти є основною цільовою групою. Вони використовують систему для вивчення лінійної алгебри та аналітичної геометрії, виконання навчальних

завдань, візуалізації операцій над векторами та самостійного опрацювання матеріалу.

Таблиця 2.1 – Категорії користувачів системи

Категорія	**Опис**	**Типові сценарії використання**
Студенти	Особи, що вивчають лінійну алгебру та аналітичну геометрію	Виконання навчальних завдань, візуалізація операцій, самостійне навчання
Викладачі	Викладачі закладів вищої освіти	Демонстрація матеріалу, пояснення тем, підготовка прикладів
Інженери	Фахівці технічних спеціальностей	Перевірка розрахунків, швидке моделювання
Дослідники	Науковці та аспіранти	Аналіз геометричних об’єктів, експериментальні обчислення

Викладачі застосовують програму як демонстраційний інструмент під час занять. Вона допомагає пояснювати теоретичний матеріал, показувати приклади та готувати наочні навчальні матеріали.

Інженери використовують систему для швидких розрахунків і перевірки результатів у прикладних задачах, а також для моделювання векторних операцій.

Дослідники застосовують інструмент для аналізу геометричних об’єктів, перевірки гіпотез і виконання експериментальних обчислень.

Функціональні вимоги визначають перелік можливостей, які повинна забезпечувати система для реалізації поставлених цілей. Вони структуровані за основними функціональними напрямами: візуалізація, математичні операції, інтерфейс та навчальні функції.

У таблиці 2.2 наведено деталізовану специфікацію функціональних вимог із зазначенням їх пріоритетності. Це дозволяє розмежувати критично важливі функції (без яких система не може працювати) та додаткові можливості, що розширюють функціонал.

Таблиця містить узагальнену специфікацію функціональних вимог до розроблюваної системи “ВекторВіз”. У ній кожна вимога має унікальний ідентифікатор (ID), що дозволяє однозначно її ідентифікувати та використовувати під час проєктування і реалізації програмного засобу. Вимоги

згруповані за модулями (візуалізація, операції над векторами, аналіз, інтерфейс та навчальні функції), що відображає структурну організацію системи.

Таблиця 2.2 – Функціональні вимоги до системи

ID	Модуль	Вимога	Пріоритет
FR-001	Візуалізація	Відображення векторів у 2D просторі	Критичний
FR-002	Візуалізація	Відображення векторів у 3D просторі з проекцією	Критичний
FR-003	Візуалізація	Масштабування області перегляду	Високий
FR-004	Візуалізація	Панорамування (переміщення) полотна	Високий
FR-005	Візуалізація	Обертання 3D сцени мишею	Високий
FR-006	Візуалізація	Відображення координатної сітки	Середній
FR-007	Візуалізація	Відображення осей координат з підписами	Середній
FR-008	Візуалізація	Анімація операцій над векторами	Середній
FR-009	Операції	Додавання векторів	Критичний
FR-010	Операції	Віднімання векторів	Критичний
FR-011	Операції	Множення вектора на скаляр	Критичний
FR-012	Операції	Обчислення скалярного добутку	Критичний
FR-013	Операції	Обчислення векторного добутку	Критичний
FR-014	Операції	Обчислення довжини (норми) вектора	Високий
FR-015	Операції	Нормалізація вектора	Високий
FR-016	Операції	Обчислення кута між векторами	Високий
FR-017	Операції	Проекція вектора на вектор	Високий
FR-018	Операції	Обертання вектора на заданий кут	Середній
FR-019	Аналіз	Перевірка колінеарності векторів	Середній
FR-020	Аналіз	Перевірка ортогональності векторів	Середній
FR-021	Аналіз	Обчислення площі паралелограма	Середній
FR-022	Аналіз	Обчислення об’єму паралелепіпеда	Середній
FR-023	Аналіз	Ортогоналізація Грама-Шмідта	Низький
FR-024	UI	Введення координат векторів	Критичний
FR-025	UI	Вибір кольору векторів	Середній
FR-026	UI	Перемикання 2D/3D режимів	Високий
FR-027	UI	Експорт зображення	Низький
FR-028	UI	Збереження/завантаження проєкту	Низький
FR-029	Навчання	Інтерактивні підказки	Середній
FR-030	Навчання	Покроковий режим операцій	Середній

Також у таблиці визначено пріоритет кожної функції, що дозволяє розмежувати критично необхідний базовий функціонал і додаткові можливості, які розширюють систему. Окремо зазначено статус реалізації, який показує, що всі вимоги були реалізовані на етапі розробки програмного продукту. Загалом таблиця забезпечує повне уявлення про функціональні можливості системи та слугує основою для подальшого проєктування й тестування.

Окрім функціональних можливостей, було визначено нефункціональні вимоги (табл. 2.3), які характеризують якість роботи системи. Вони описують такі аспекти, як продуктивність, надійність, зручність використання та сумісність.

Таблиця 2.3 – Нефункціональні вимоги

ID	Категорія	Вимога	Метрика
NFR-001	Продуктивність	Час відгуку на дії користувача	< 100 мс
NFR-002	Продуктивність	Частота оновлення анімації	≥ 30 FPS
NFR-003	Продуктивність	Максимальна кількість векторів	≥ 100
NFR-004	Надійність	Коректність обчислень	Похибка < 10^{-10}
NFR-005	Надійність	Стабільність роботи	Без критичних збоїв
NFR-006	Зручність	Час навчання базовим функціям	< 15 хвилин
NFR-007	Зручність	Інтуїтивність інтерфейсу	SUS ≥ 70
NFR-008	Портативність	Кросплатформність	Windows, Linux, macOS
NFR-009	Портативність	Мінімальні залежності	Лише стандартна бібліотека Python
NFR-010	Підтримуваність	Модульна архітектура	Coupling < 0.3

Зокрема, система повинна забезпечувати швидку реакцію на дії користувача (менше 100 мс), плавну анімацію з частотою не менше 30 кадрів на секунду, а також стабільну роботу без критичних збоїв. Важливою вимогою є також інтуїтивно зрозумілий інтерфейс, що дозволяє користувачу освоїти основні функції за короткий проміжок часу.

Крім того, система повинна бути кросплатформною та працювати на основних операційних системах без додаткових складних налаштувань, що забезпечує її доступність для широкого кола користувачів.

Варіанти використання системи «ВекторВіз» відображають сукупність основних сценаріїв взаємодії користувача з програмним засобом та визначають, які саме функції доступні в межах навчального середовища. Усі сценарії

побудовані навколо одного базового актора – користувача, який може виступати як студент, викладач або інший зацікавлений у вивченні векторної алгебри користувач.

Основні варіанти використання охоплюють створення, редагування та видалення векторів, що формують вихідні дані для подальших математичних операцій. На основі введених векторів користувач виконує обчислювальні дії, зокрема додавання та віднімання векторів, множення на скаляр, обчислення скалярного добутку, визначення довжини вектора, кута між векторами та інших геометричних характеристик.

Результати виконаних операцій одразу відображаються у графічному вигляді на координатній площині, що забезпечує наочність математичних перетворень. Окрім цього, користувач має можливість змінювати режим відображення (2D/3D), масштабувати та переміщувати робочу область, а також налаштовувати візуальні параметри елементів сцени.

Окремий блок функціональності становлять допоміжні та навчальні можливості системи. До них належить експорт графічного зображення, збереження та завантаження проєктів, перегляд довідкових матеріалів, а також аналіз властивостей векторів. Ці функції забезпечують підтримку як навчального процесу, так і самостійної роботи користувача, підвищуючи загальну ефективність використання програмного засобу.

Ключовим сценарієм використання системи є виконання математичних операцій над векторами, оскільки саме він поєднує обчислювальну, аналітичну та візуальну складові застосунку (табл. 2.4).

У межах цього сценарію користувач обирає необхідну операцію через інтерфейс програми. Далі система перевіряє наявність достатньої кількості векторів та коректність введених координат. Після цього виконується відповідне математичне обчислення, результат якого формується у числовому вигляді.

Паралельно система будує геометричну інтерпретацію отриманого результату, відображаючи його на координатній площині. Це дозволяє користувачу не лише отримати числову відповідь, але й побачити її візуальне

представлення. Додатково виводиться текстове пояснення виконаної операції, що сприяє кращому засвоєнню навчального матеріалу.

Таблиця 2.4 – Специфікація UC4 «Виконати операцію над векторами»

Елемент	Опис
Назва	Виконання математичної операції над векторами
Актор	Користувач
Передумови	У системі створено щонайменше один або два вектори залежно від типу операції
Основний сценарій	1. Користувач обирає операцію (додавання, віднімання тощо) 2. Система перевіряє коректність вхідних даних 3. Виконується математичне обчислення 4. Формується результат у числовому вигляді 5. Результат візуалізується на координатній площині 6. Відображається пояснення виконаної операції
Альтернативний сценарій	У разі некоректного введення система запитує повторне введення даних
Виключні ситуації	Недостатня кількість векторів або помилки введення координат
Постумови	Отриманий результат відображено у графічному та числовому вигляді

У випадку некоректного введення даних або недостатньої кількості векторів система формує повідомлення про помилку та запитує повторне введення інформації. Після успішного завершення операції результат може бути збережений у межах проєкту або використаний для подальших обчислень.

Отже, сформовано набір функціональних вимог до програмного засобу, що охоплюють візуалізацію векторів, виконання основних математичних операцій та допоміжні навчальні можливості.

Визначено також категорії користувачів і пріоритетність реалізації функцій, що дозволило структурувати розвиток системи та уточнити її призначення. Отримані результати стали основою для подальшого проєктування архітектури програми.

2.2 Структурне проєктування системи

Структурне проєктування програмного засобу «ВекторВіз» передбачає визначення архітектурної організації системи, виділення основних підсистем та встановлення зв’язків між ними. Метою цього етапу є формування логічної та масштабованої структури програмного продукту, яка забезпечує розділення відповідальностей, зручність супроводу та можливість подальшого розширення функціоналу.

Система побудована на основі архітектурного патерну Model–View–Controller (MVC) [22], який є класичним підходом для розробки інтерактивних застосунків (рис. 2.1). Його використання дозволяє чітко розділити логіку представлення даних, бізнес-логіку та керування взаємодією користувача з системою.

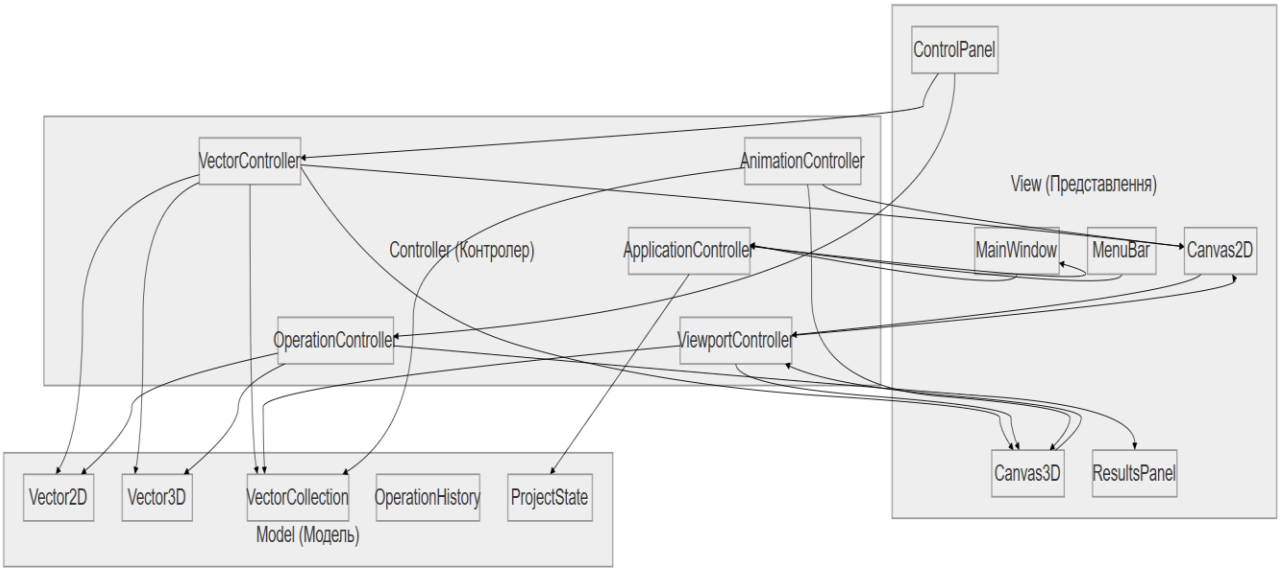


Рисунок 2.1 – Архітектурний патерн

Рівень View (представлення) відповідає за графічний інтерфейс користувача. Він включає головне вікно застосунку, 2D та 3D полотна для візуалізації векторів, панель керування, панель результатів та меню. Саме цей

рівень забезпечує взаємодію користувача з програмою та відображення всіх обчислень у візуальній формі.

Рівень Controller (контролер) виконує обробку подій та координує роботу між інтерфейсом і моделлю даних. До його складу входять головний контролер застосунку, контролер векторів, контролер операцій, контролер області перегляду та контролер анімації. Він забезпечує передачу команд від користувача до відповідних модулів системи.

Рівень Model (модель) містить основні структури даних та математичну логіку. Сюди входять класи, що описують вектори у двовимірному та тривимірному просторі, колекцію векторів, історію операцій та стан проєкту. Саме цей рівень відповідає за виконання математичних обчислень та збереження даних.

Архітектура системи додатково деталізується через виділення основних програмних шарів: презентаційного, бізнес-логіки, рівня даних та утиліт (рис. 2.2).

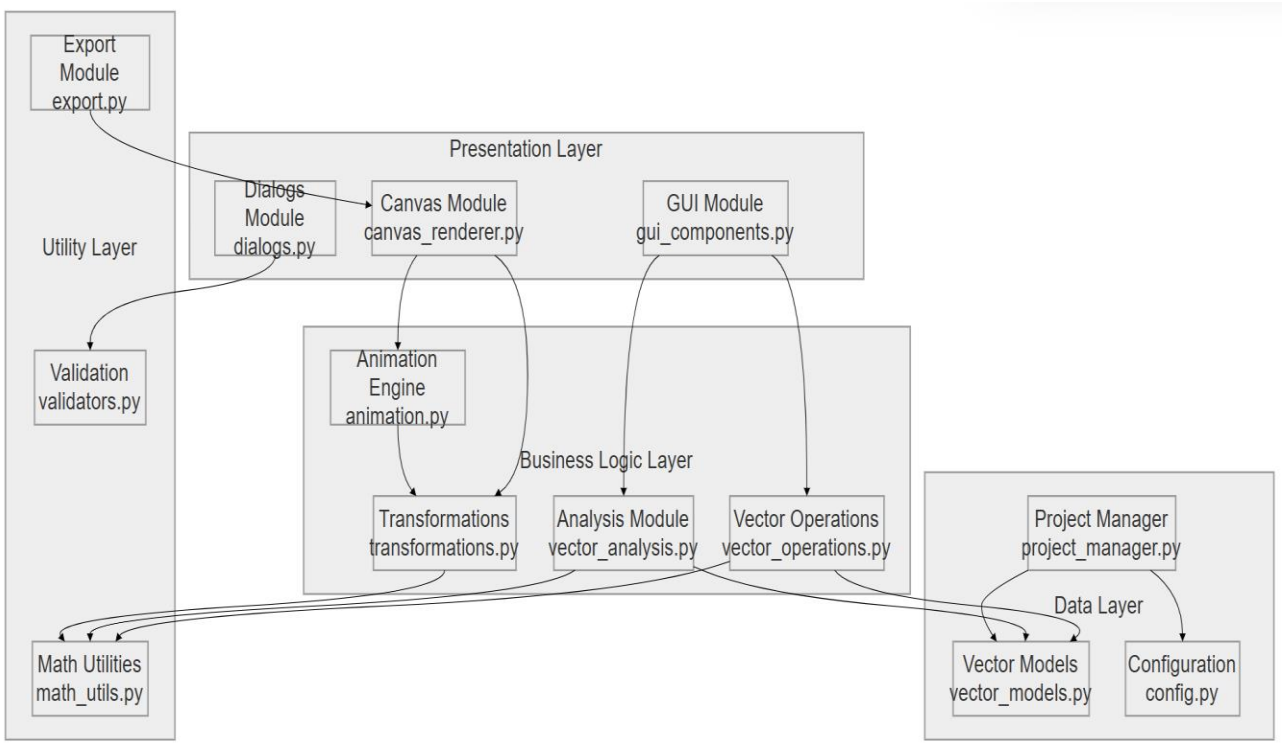


Рисунок 2.2 – Діаграма компонентів

Презентаційний рівень включає модулі графічного інтерфейсу, канвас для відображення графіки та діалогові вікна. Він відповідає за взаємодію з користувачем і відображення результатів роботи системи.

Рівень бізнес-логіки містить модулі виконання математичних операцій над векторами, аналізу їх властивостей, а також механізм анімації та геометричних перетворень. Саме тут реалізується основна функціональність системи.

Рівень даних включає моделі векторів, менеджер проєктів та конфігураційні налаштування. Він забезпечує збереження та структурування інформації.

Утилітарний рівень містить допоміжні модулі для математичних розрахунків, експорту результатів та перевірки коректності введених даних.

Структура класів (рис. 2.3) відображає об’єктно-орієнтовану модель системи та визначає основні сутності, які використовуються для реалізації функціоналу.

Базовими класами є Vector2D та Vector3D, які описують математичні вектори у двовимірному та тривимірному просторі. Вони містять методи для обчислення довжини, нормалізації, скалярного та векторного добутку, а також геометричних перетворень, таких як обертання та проекція.

Клас VectorCollection використовується для зберігання та управління набором векторів, забезпечуючи операції додавання, видалення та доступу до елементів.

Клас VectorOperations реалізує базові математичні операції над векторами, включаючи додавання, віднімання, множення на скаляр, обчислення скалярного та векторного добутку, а також кутів між векторами.

Клас VectorAnalyzer призначений для аналізу геометричних властивостей векторів, таких як колінеарність, ортогональність, площа та об’єм відповідних геометричних фігур, а також ортогоналізація системи векторів.

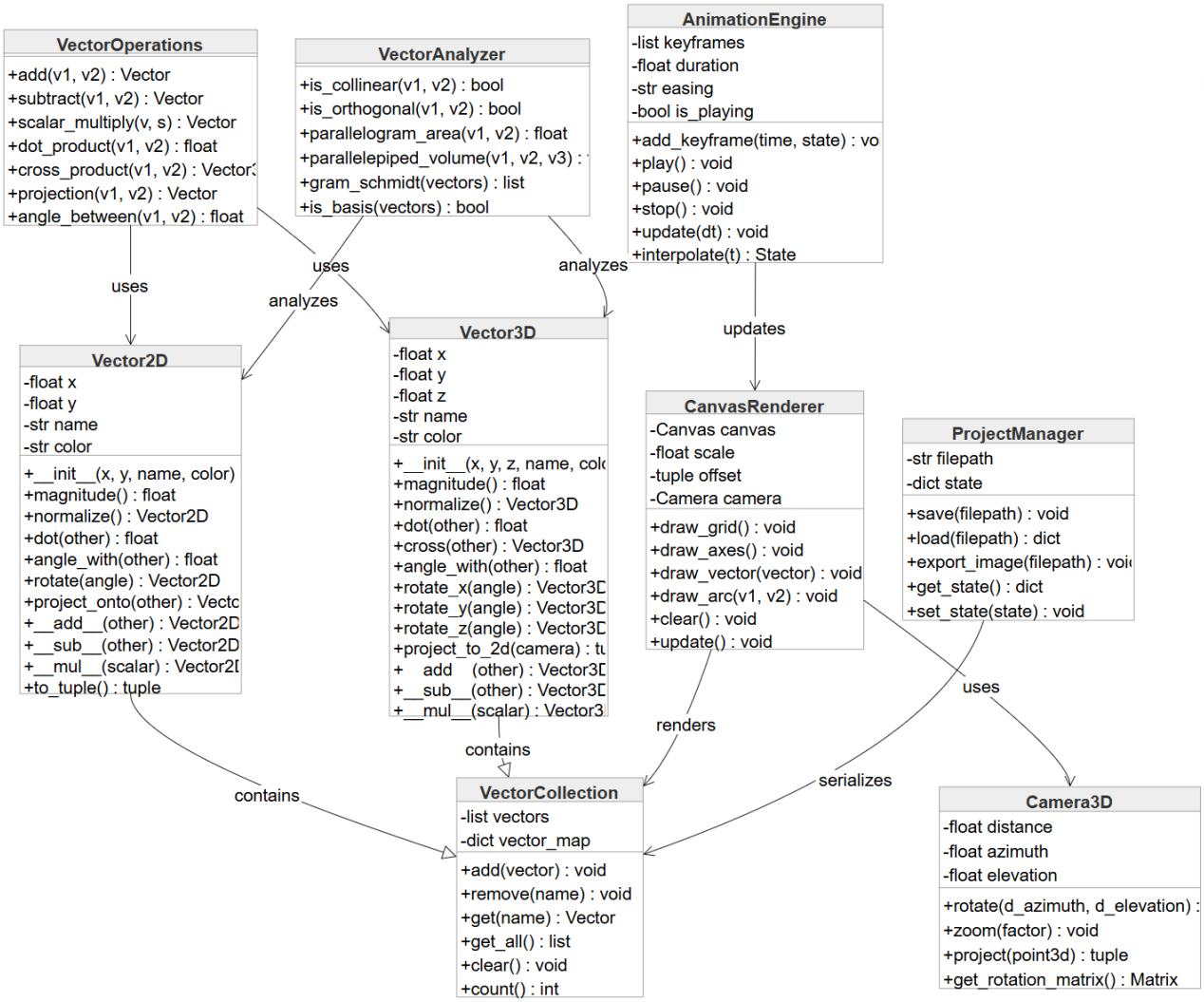


Рисунок 2.3 – Структура класів

Клас CanvasRenderer відповідає за графічне відображення векторів, координатної сітки та допоміжних елементів на канвасі, а також взаємодіє з камерою для 3D-проекцій.

Клас Camera3D реалізує механізм тривимірної візуалізації, включаючи обертання сцени, масштабування та проекцію 3D-координат у 2D-площину.

Клас AnimationEngine забезпечує анімацію змін стану системи, використовуючи ключові кадри та інтерполяцію між ними.

Клас ProjectManager відповідає за збереження, завантаження та експорт стану проєкту, включаючи серіалізацію даних та експорт зображень.

Для забезпечення модульності, розширюваності та зручності супроводу програмного засобу його функціональність була розподілена на окремі

програмні модулі. Кожен модуль відповідає за виконання певного набору завдань та взаємодіє з іншими компонентами через визначені інтерфейси. Такий підхід дозволяє зменшити залежність між компонентами системи та спростити подальшу модернізацію програмного забезпечення.

Основу системи становить модуль Core, який реалізує базові класи для роботи з двовимірними та тривимірними векторами. Саме цей модуль забезпечує зберігання координат, виконання базових математичних операцій та надання даних іншим підсистемам програми.

Модуль Operations реалізує основні математичні операції над векторами, зокрема додавання, віднімання, множення на скаляр, обчислення скалярного та векторного добутків, нормалізацію та знаходження проекцій. Оскільки всі операції виконуються над об’єктами класів векторів, модуль залежить від математичного ядра системи.

Для реалізації додаткових математичних досліджень призначено модуль Analysis, який забезпечує перевірку колінеарності та ортогональності векторів, обчислення площі паралелограма, об’єму паралелепіпеда та виконання ортогоналізації за методом Грама–Шмідта.

Візуальне представлення результатів роботи користувача реалізується за допомогою модуля Rendering, який відповідає за побудову координатної сітки, осей координат, векторів, допоміжних геометричних елементів та відображення результатів обчислень на полотні Tkinter Canvas.

Оскільки програмний засіб підтримує роботу з тривимірними об’єктами, було розроблено окремий модуль Camera, який виконує проєкцію тривимірних координат на площину екрана, а також забезпечує масштабування та обертання сцени.

Для підвищення наочності навчального процесу використовується модуль Animation, який відповідає за анімоване відображення математичних операцій та плавну зміну станів графічних об’єктів.

Модуль Transform реалізує геометричні перетворення, зокрема перенесення, масштабування та обертання векторів і координатних систем.

Графічний інтерфейс користувача реалізовано у модулі GUI, який містить головне вікно програми, панелі керування, меню та елементи взаємодії з користувачем. Для реалізації діалогових вікон використовується окремий модуль Dialogs.

Модулі Project та Export забезпечують збереження результатів роботи користувача. Перший відповідає за збереження та завантаження проєктів у форматі JSON, а другий – за експорт графічних результатів у вигляді зображень.

Допоміжні функції зосереджені у модулях Config, Utils та Validation, які забезпечують зберігання налаштувань програми, виконання математичних обчислень та перевірку коректності введених даних відповідно.

Опис основних модулів програмного засобу наведено в таблиці 2.5.

Таблиця 2.5 – Опис модулів системи

Модуль	**Файл**	**Відповідальність**	**Залежності**
Core	`vector_models.py`	Базові класи Vector2D, Vector3D	math
Operations	`vector_operations.py`	Математичні операції над векторами	Core
Analysis	`vector_analysis.py`	Аналітичні функції	Core, Operations
Rendering	`canvas_renderer.py`	Відображення на Canvas	Core, tkinter
Camera	`camera.py`	3D проєкція та керування камерою	math
Animation	`animation.py`	Анімаційний рушій	Rendering
Transform	`transformations.py`	Геометричні трансформації	Core, math
GUI	`gui_components.py`	Елементи інтерфейсу	tkinter
Dialogs	`dialogs.py`	Діалогові вікна	GUI
Project	`project_manager.py`	Збереження та завантаження проєктів	Core, json
Export	`export.py`	Експорт зображень	Rendering, PIL
Config	`config.py`	Конфігурація програми	json
Utils	`math_utils.py`	Допоміжні математичні функції	math
Validation	`validators.py`	Валідація введення даних	–
Main	`main.py`	Точка входу та ініціалізація системи	Усі модулі

Для організації вихідного коду та спрощення процесу розробки програмний засіб було структуровано у вигляді окремих пакетів. Така організація дозволяє логічно групувати компоненти за їхнім функціональним

призначенням та мінімізувати взаємозалежності між різними частинами системи.

До складу пакета core входять модулі, що реалізують математичне ядро системи. Тут зосереджено класи представлення векторів, модулі виконання математичних операцій та аналітичних розрахунків.

Пакет visualization забезпечує графічне відображення об’єктів. До нього належать компоненти побудови сцени, керування камерою та анімації математичних процесів.

Пакет ui містить усі елементи користувацького інтерфейсу: головні вікна, панелі керування, діалогові вікна та допоміжні віджети.

Для роботи з файлами використовується пакет io, до складу якого входять модулі збереження проєктів, експорту результатів та керування конфігураційними параметрами системи.

Допоміжний пакет utils забезпечує виконання службових операцій, математичних обчислень, перевірку даних та реалізацію загальних функцій, які використовуються різними компонентами системи.

Запропонована структура пакетів сприяє підвищенню читабельності програмного коду, полегшує тестування окремих компонентів та забезпечує можливість подальшого розширення функціональності програмного засобу.

Діаграма послідовності (рис. 2.6) описує процес взаємодії між основними компонентами системи під час виконання операції додавання векторів. Спочатку користувач обирає відповідну команду в графічному інтерфейсі. Після цього інтерфейс передає запит до контролера векторів, який отримує вибрані об’єкти та передає їх до модуля математичних операцій.

Модуль VectorOperations виконує обчислення суми векторів шляхом виклику відповідного методу класу вектора. Отриманий результат повертається до контролера, який ініціює запуск анімації операції. Під час відтворення анімації модуль Animation формує проміжні стани об’єктів та передає їх модулю візуалізації для відображення на полотні.

Після завершення анімації контролер оновлює графічне представлення результату та передає числові дані до панелі результатів. У підсумку користувач отримує як графічну інтерпретацію операції, так і її числовий результат.

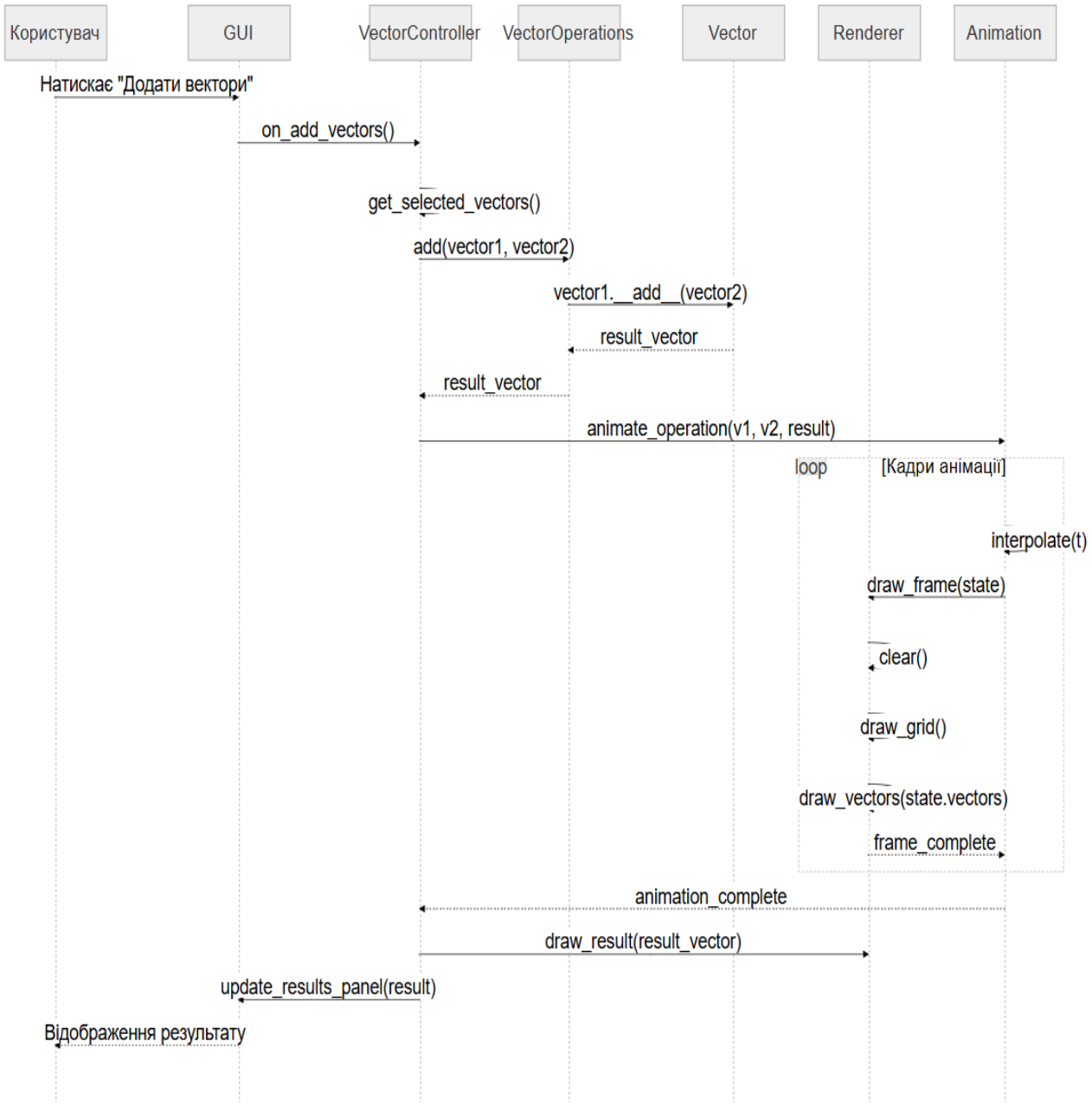


Рисунок 2.6 – Діаграма послідовності

Для опису поведінки застосунку під час роботи користувача було побудовано діаграму станів (рис. 2.7). Початковим станом системи є режим очікування після запуску програми. Із цього стану користувач може створювати

нові вектори, виконувати математичні операції, змінювати параметри відображення або працювати з файлами проєктів.

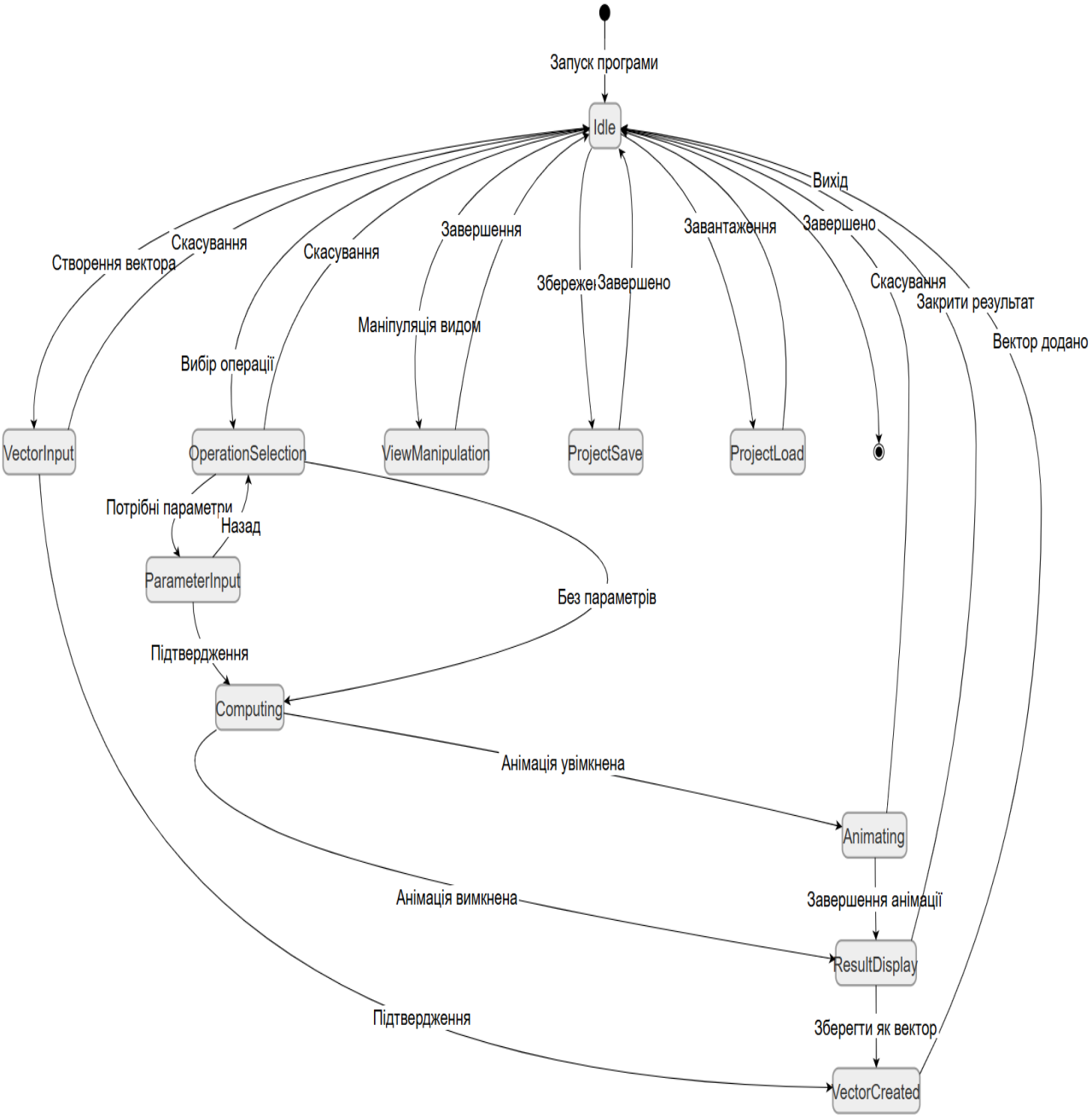


Рисунок 2.7 – Діаграма станів

Після створення вектора система переходить до стану його збереження та подальшого відображення на полотні. У разі виконання математичної операції програма переходить до стану обчислення, після чого результат може бути безпосередньо відображений або анімований.

Окремі стани передбачені для керування видом візуалізації, збереження та завантаження проєктів. Після завершення будь-якої операції система повертається до базового режиму очікування, що забезпечує можливість безперервної роботи користувача з програмою.

Графічний інтерфейс програмного засобу розроблено відповідно до принципів зручності використання та швидкого доступу до основних функцій. Центральним елементом інтерфейсу є головне вікно MainWindow, яке об’єднує всі функціональні компоненти системи.

У верхній частині вікна розташоване головне меню, що містить розділи для роботи з файлами, редагування об’єктів, налаштування відображення, виконання операцій та перегляду довідкової інформації.

Під меню знаходиться панель інструментів, яка забезпечує швидкий доступ до найбільш використовуваних функцій, таких як створення та видалення векторів, масштабування сцени та перемикання між двовимірним і тривимірним режимами.

Основна область інтерфейсу поділена на три частини. Ліва панель використовується для відображення списку створених векторів та їхніх властивостей. Центральна область містить полотно візуалізації, на якому відображаються результати математичних операцій, а також рядок стану з поточною службовою інформацією. Права панель призначена для вибору операцій, перегляду результатів обчислень та відображення додаткових відомостей про вибрані об’єкти.

Така структура забезпечує логічне розміщення елементів інтерфейсу та підвищує ефективність взаємодії користувача із програмним засобом.

Для забезпечення взаємодії між окремими компонентами системи були визначені програмні інтерфейси основних модулів. Найважливішими серед них є модулі VectorOperations та CanvasRenderer, які відповідають відповідно за математичні обчислення та графічне відображення результатів.

Інтерфейс модуля VectorOperations містить набір методів для виконання базових операцій над векторами. До них належать додавання, віднімання,

множення на скаляр, обчислення скалярного та векторного добутків, визначення кута між векторами, знаходження проєкції та нормалізація. Усі методи приймають на вхід векторні об’єкти та повертають результат обчислень у відповідному форматі.

Інтерфейс модуля CanvasRenderer забезпечує побудову графічних елементів на полотні. До його функцій належать відображення координатної сітки та осей координат, побудова векторів і геометричних фігур, зміна масштабу та координатних перетворень. Крім того, модуль виконує перетворення між математичною системою координат та координатами екрана, що є необхідним для коректної візуалізації об’єктів.

Отже, у результаті структурного проєктування було сформовано архітектуру навчального програмного засобу для інтерактивної візуалізації векторної алгебри. Визначено архітектурний патерн MVC, структуру основних модулів, класів та пакетів системи, а також описано механізми взаємодії між її компонентами. Запропонована структура забезпечує модульність, розширюваність, зручність супроводу та ефективну реалізацію функціональних можливостей програмного засобу.

2.3 Розробка алгоритмів візуалізації векторів та операцій над ними

Одним із ключових завдань під час розробки програмного засобу є створення алгоритмів, які забезпечують коректне відображення математичних об’єктів на екрані та реалізують наочну демонстрацію результатів операцій над векторами. Оскільки користувач працює з математичними координатами, а графічне відображення здійснюється в піксельній системі координат екрана, необхідно забезпечити перетворення між цими двома просторами.

Розроблені алгоритми повинні забезпечувати точність побудови геометричних об’єктів, підтримку масштабування та переміщення області перегляду, а також можливість інтерактивної взаємодії користувача з векторами.

Крім того, важливим аспектом є забезпечення наочності відображення математичних операцій шляхом використання анімації та допоміжних графічних елементів.

Основою роботи системи візуалізації є алгоритм перетворення математичних координат у координати графічного полотна (рис. 2.8). У математичній системі координат початок відліку розташовується в центрі координатної площини, при цьому вісь Y спрямована вгору. У графічній системі координат Tkinter початок координат знаходиться у верхньому лівому куті вікна, а вісь Y спрямована вниз.



Рисунок 2.8 – Алгоритм перетворення математичних координат у координати графічного полотна

Для узгодження цих двох систем використовується спеціальне афінне перетворення, яке враховує поточний масштаб відображення та зміщення

області перегляду. Коефіцієнт масштабування визначає кількість пікселів, що відповідають одній математичній одиниці, а параметри зміщення дозволяють реалізувати панорамування сцени.

У процесі побудови графічного об’єкта система отримує координати точки у світовій системі координат, після чого виконує масштабування, інвертування осі Y та перенесення початку координат до центру полотна. Отримані значення використовуються для відображення відповідного елемента на екрані.

Зворотне перетворення використовується під час взаємодії користувача з об’єктами. Наприклад, при перетягуванні вектора мишею система повинна визначити відповідні математичні координати точки, на яку вказує курсор. Для цього застосовується обернена формула перетворення.

Використання такого підходу забезпечує незалежність математичної моделі від параметрів графічного інтерфейсу та дозволяє реалізувати масштабування без втрати точності розрахунків.

Після перетворення координат виконується побудова вектора на полотні. У системі кожен вектор подається як напрямлений відрізок, що складається з основної лінії та наконечника стрілки.

Алгоритм побудови починається з визначення початкової та кінцевої точок вектора (рис. 2.9). У більшості випадків початкова точка збігається з початком координат, а кінцева визначається координатами самого вектора. Далі обидві точки перетворюються із математичної системи координат у координати екрана.

Після отримання екранних координат виконується перевірка довжини вектора. Якщо довжина менша за встановлений поріг, його відображення не здійснюється, оскільки надто короткі об’єкти можуть погіршувати сприйняття графічної сцени.

Для формування наконечника стрілки обчислюється кут нахилу вектора відносно осі X. На основі цього кута визначаються координати двох додаткових точок, які формують бокові сторони наконечника. Отримані координати використовуються для побудови завершеної стрілки.



Рисунок 2.9 – Алгоритм побудови

Важливою особливістю алгоритму є автоматичне відображення підпису вектора. Назва розміщується поруч із кінцевою точкою та зміщується на певну відстань для уникнення накладання тексту на графічні елементи.

Такий підхід дозволяє забезпечити зрозуміле та наочне відображення математичних об’єктів незалежно від їхніх координат та масштабу сцени.

Для побудови наконечника використовуються тригонометричні співвідношення, які дозволяють визначити координати його бокових точок відносно кінця вектора. Використання функції arctan2 забезпечує коректне визначення кута нахилу у всіх квадрантах координатної площини.

Основні параметри алгоритму відображення вектора наведено в таблиці 2.8.

Таблиця 2.8 – Параметри відображення вектора

Параметр	Значення за замовчуванням	Опис
`arrow_length`	15 px	Довжина наконечника
`arrow_angle`	25°	Кут розкриття наконечника
`line_width`	2 px	Товщина лінії
`label_offset`	10 px	Відступ підпису
`min_length`	5 px	Мінімальна довжина для відображення

Значення параметрів було підібрано експериментально з урахуванням забезпечення достатньої наочності та читабельності графічних об’єктів на екрані. Вони можуть бути змінені залежно від масштабу сцени або налаштувань користувача без зміни основного алгоритму відображення.

Для забезпечення візуалізації просторових векторів у програмі реалізовано механізм тривимірної проекції, який дозволяє відображати об’єкти тривимірного простору на двовимірному полотні екрана. Основою алгоритму є використання матриць обертання та ортографічної проекції. Такий підхід забезпечує наочне відображення взаємного розташування векторів у просторі та дозволяє користувачеві змінювати ракурс перегляду шляхом обертання віртуальної камери.

Перед відображенням кожна просторова точка проходить процедуру трансформації відповідно до положення камери. Для цього використовуються три кути орієнтації: азимут θ, який визначає поворот навколо вертикальної осі; кут підйому φ, що задає нахил камери відносно горизонтальної площини; та кут нахилу ψ, який використовується для додаткового обертання навколо осі спостереження.

Загальна матриця повороту формується як добуток трьох елементарних матриць обертання:

$$R = R_z(\psi) \cdot R_x(\phi) \cdot R_y(\theta).$$

Після побудови матриці обертання координати точки перетворюються відповідно до нового положення камери:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = R \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Отримані координати використовуються для побудови двовимірної проекції. Оскільки відображення здійснюється на плоскому екрані, координата z' не використовується безпосередньо при побудові зображення, а значення x' та y' масштабуються відповідно до поточного масштабу сцени:

$$x_s = x' \cdot S + C_x,$$
$$y_s = C_y - y' \cdot S,$$

де S – коефіцієнт масштабування, а C_x та C_y визначають положення центру полотна.

Процес виконання алгоритму наведено на рисунку 2.10.



Рисунок 2.10 – Алгоритм 3D проекції

Діаграма 2.10 відображає послідовність виконання алгоритму 3D проекції точки. Спочатку система отримує координати просторової точки та параметри орієнтації камери (кути θ, φ, ψ). Далі поетапно формуються матриці обертання для кожної осі та обчислюється загальна матриця трансформації. Після цього виконується перетворення вихідної 3D точки у повернуту систему координат. Отримані значення масштабуються та переводяться в екранні координати, що дозволяє відобразити тривимірний об’єкт на двовимірному полотні.

Запропонований алгоритм забезпечує коректне відображення тривимірних об’єктів та створює можливість інтерактивного перегляду сцени під різними кутами, що є важливим для вивчення просторових операцій над векторами.

Для підвищення наочності та покращення сприйняття результатів обчислень у програмному засобі реалізовано механізм анімації. Його основним завданням є плавне відображення переходу між початковим та кінцевим станами об’єктів. Анімація дозволяє користувачеві спостерігати процес виконання операції, а не лише отримувати її результат.

Робота алгоритму починається із фіксації початкового стану сцени S_0 та визначення кінцевого стану S_1. Після цього встановлюється тривалість анімації та запускається таймер. На кожному кроці обчислюється відносний прогрес виконання:

$$p = \frac{t}{T},$$

де t – поточний момент часу, а T – загальна тривалість анімації.

Для створення природного руху використовується функція згладжування (easing), яка модифікує значення прогресу. У системі реалізовано кілька типів згладжування: лінійне, ease-in-out, ease-out та bounce. Вибір функції визначає характер руху анімованого об’єкта.

Після обчислення скоригованого прогресу виконується інтерполяція між початковим і кінцевим станами сцени. Для цього використовується лінійна інтерполяція:

$$\vec{V}(t) = (1-t)\cdot\vec{V}_0 + t\cdot\vec{V}_1.$$

У випадках, коли необхідно плавно змінювати напрямок вектора, застосовується сферична лінійна інтерполяція (SLERP):

$$\vec{V}(t) = \frac{\sin((1-t)\Omega}{sin\Omega}\vec{V}_0 + \frac{\sin(t\Omega)}{sin\Omega}\vec{V}_1,$$

де $\Omega = arccos(\vec{V}_0 \cdot \vec{V}_1)$

На основі отриманих проміжних значень виконується відображення чергового кадру анімації. Процес повторюється до завершення встановленого часу.

Загальна послідовність виконання алгоритму наведена на рис. 2.11.

Рисунок 2.11 ілюструє алгоритм анімації переходу між станами об’єкта. Спочатку фіксуються початковий і кінцевий стани та задається тривалість анімації. Далі запускається часовий цикл, у межах якого обчислюється прогрес виконання та застосовується функція згладжування (easing) для забезпечення плавності руху. На кожному кроці виконується інтерполяція між станами та оновлення кадру. Процес повторюється до завершення часу, після чого встановлюється фінальний стан.

Застосування анімації дозволяє зробити процес вивчення векторної алгебри більш наочним та інтуїтивно зрозумілим, оскільки користувач може спостерігати кожен етап виконання математичної операції.

Операція додавання векторів є однією з базових операцій векторної алгебри, тому її візуалізації приділено особливу увагу. У програмі використовується геометричний метод трикутника, згідно з яким другий вектор переноситься до кінця першого без зміни свого напрямку та довжини.

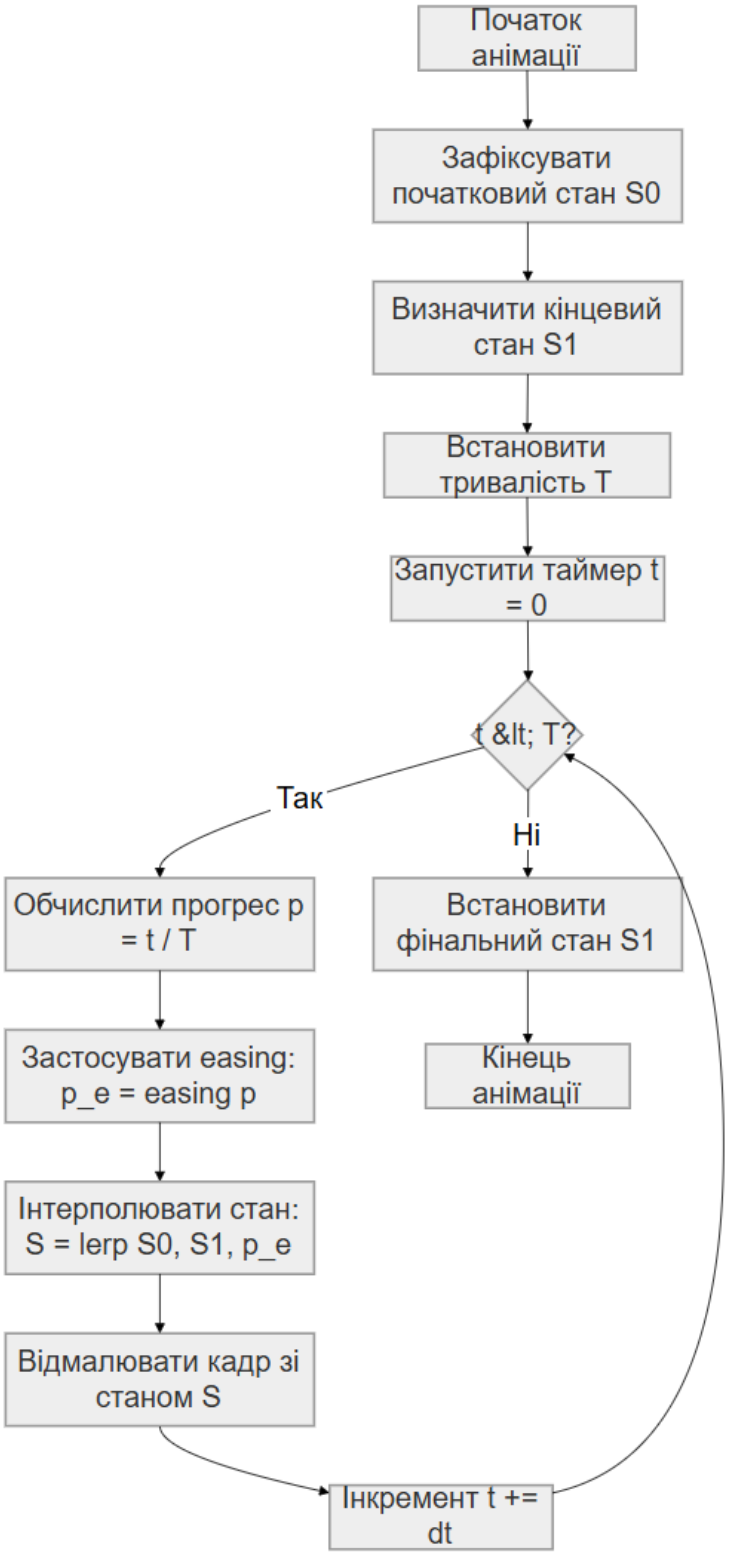


Рисунок 2.11 – Алгоритм анімації

Алгоритм складається з кількох послідовних фаз. На першому етапі відображається вектор A, який поступово з’являється на полотні за допомогою анімації. Після цього виконується відображення вектора B, який переноситься до кінцевої точки вектора A.

Після завершення побудови обох векторів система обчислює результуючий вектор:

$$\vec{C} = \vec{A} + \vec{B}$$

Далі результуючий вектор поступово відображається від початку координат до кінцевої точки побудованої геометричної конструкції. Для підвищення наочності на завершальному етапі результат підсвічується кольором та супроводжується відображенням числових характеристик.

Послідовність роботи алгоритму показана на рисунку 2.12, який відображає алгоритм візуалізації додавання двох векторів. Спочатку поетапно відображається перший вектор, після чого другий переноситься до кінця першого відповідно до геометричного правила додавання. Далі система обчислює результуючий вектор і анімовано будує його від початку координат. Завершальний етап включає візуальне підсвічування результату та відображення його числових значень, що підсилює інтерпретацію операції.

Для забезпечення плавного відображення кожен етап анімації має власну тривалість та набір візуальних ефектів, що наведено в таблиці 2.9.

Таблиця 2.9 – Фази анімації додавання

Фаза	**Тривалість**	**Опис**	**Візуальний ефект**
1	0,5 с	Відображення першого вектора	Плавна поява вектора A
2	0,7 с	Переміщення другого вектора	Перенесення B до кінця A
3	0,5 с	Побудова результуючого вектора	Поступове відображення вектора C
4	0,3 с	Виділення результату	Пульсація та зміна яскравості кольору

Такий підхід дозволяє не лише демонструвати результат обчислень, а й наочно пояснювати геометричний зміст операції додавання векторів, що суттєво підвищує ефективність навчального процесу.

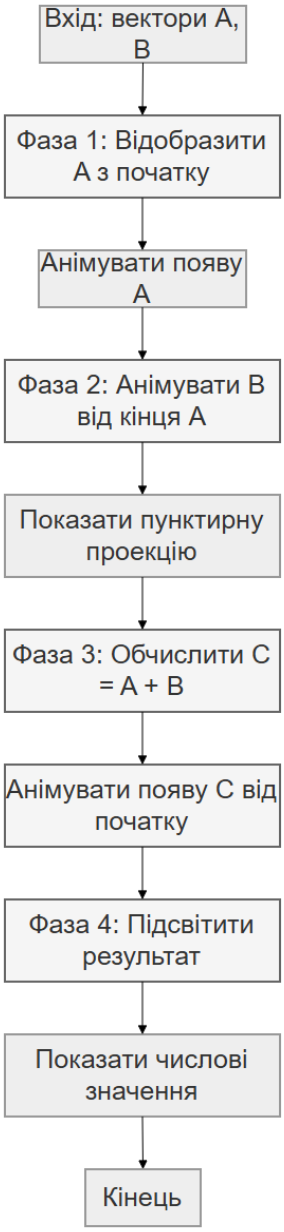


Рисунок 2.12 – Алгоритм візуалізації додавання векторів

Після розгляду базових методів побудови та перетворення геометричних об’єктів доцільно перейти до алгоритмів візуалізації конкретних математичних операцій над векторами. Однією з таких операцій є скалярний добуток (рис. 2.13), який має важливе геометричне тлумачення через кут між векторами та їх проекцію.

Подана діаграма описує поетапний процес візуалізації: від відображення двох векторів і обчислення кута між ними до побудови проекції одного вектора на інший. Далі алгоритм передбачає графічне виділення цієї проекції,

формування прямокутника площі та відображення числового результату скалярного добутку. Окремо враховується умовна візуалізація знаку результату: додатні, від’ємні або нульові значення позначаються різними кольорами, що дозволяє користувачу швидко інтерпретувати геометричний зміст операції.

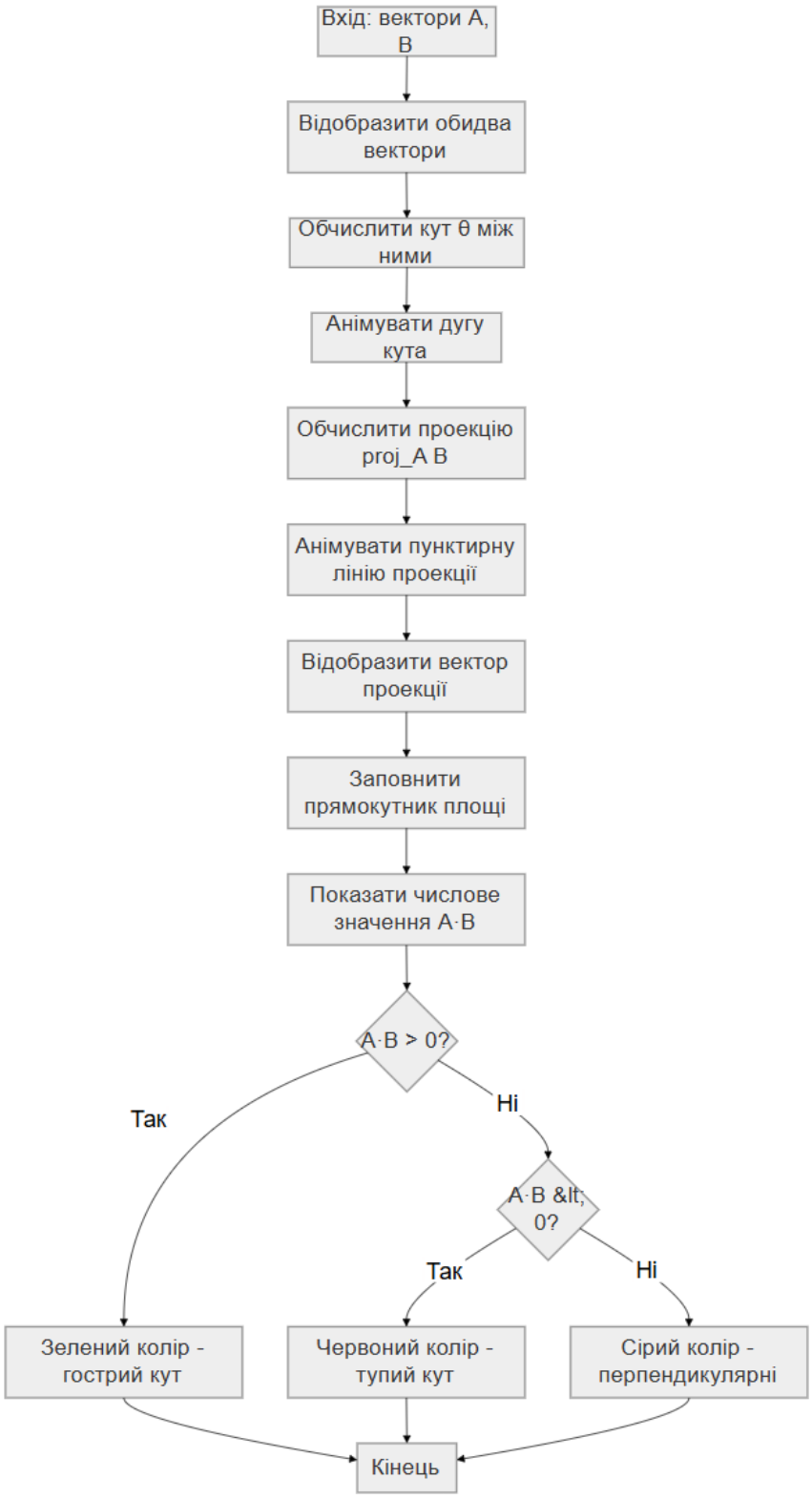


Рисунок 2.13 – Алгоритм візуалізації скалярного добутку

Наступним кроком є розгляд більш складної операції – векторного добутку, який використовується для роботи з просторовими об’єктами та визначення нормалі до площини (рис. 2.14).



Рисунок 2.14 – Алгоритм візуалізації векторного добутку

Діаграма демонструє процес переходу в 3D-режим, відображення векторів та обчислення їхнього векторного добутку. Ключовим етапом є побудова паралелограма, що утворюється двома векторами, та візуалізація його площини. Результат операції відображається у вигляді перпендикулярного вектора, який виходить із площини згідно з правилом правої руки. Додатково користувач отримує числове значення площі паралелограма, що відповідає модулю векторного добутку.

Для забезпечення зручності роботи з векторами важливим елементом є координатна сітка, яка служить опорою для візуального сприйняття простору (рис. 2.15).

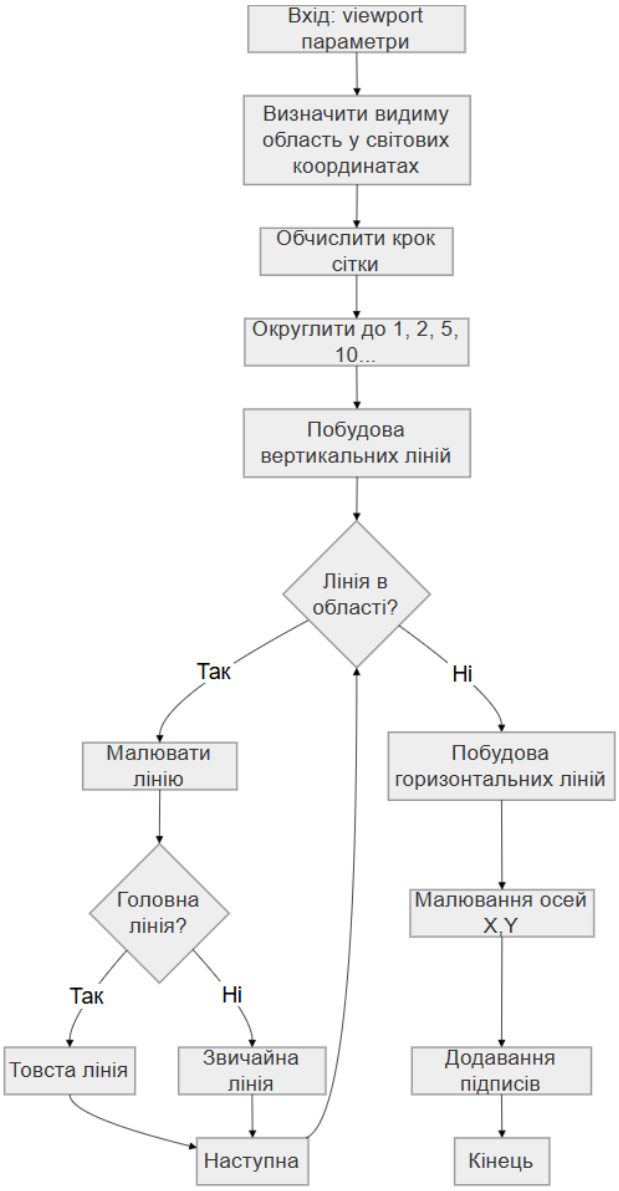


Рисунок 2.15 – Алгоритм відображення координатної сітки

Алгоритм починається з визначення видимої області відповідно до параметрів viewport і подальшого автоматичного підбору кроку сітки залежно від масштабу. Далі відбувається побудова вертикальних і горизонтальних ліній із диференційованим відображенням основних та допоміжних поділок. Окремо виділяються координатні осі, які малюються товстішими лініями та супроводжуються числовими підписами. Такий підхід забезпечує адаптивність сітки до будь-якого масштабу сцени.

Для забезпечення зручної навігації у просторі застосовується алгоритм інтерактивного масштабування з використанням колеса миші (рис. 2.16).

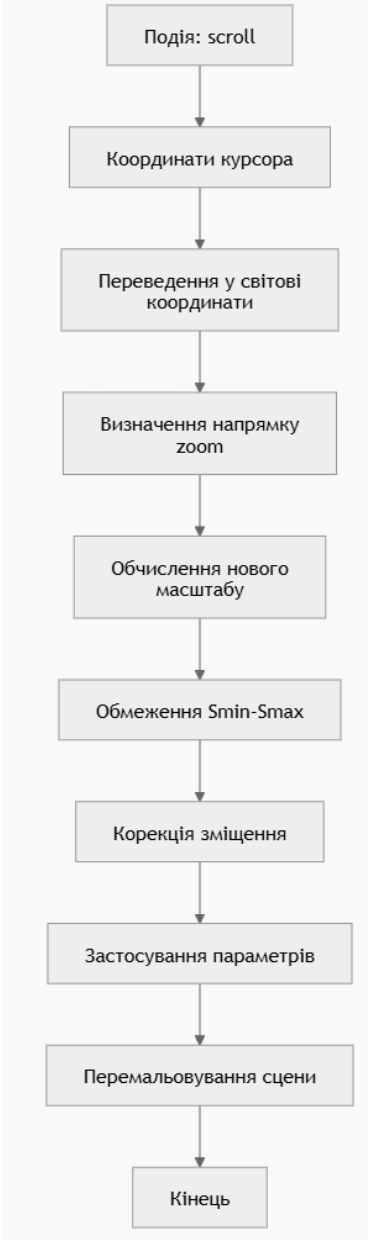


Рисунок 2.16 – Алгоритм масштабування

Алгоритм базується на зміні коефіцієнта масштабування відносно позиції курсора, що дозволяє зберігати фокус на вибраній точці. Спочатку визначається світова координата точки під курсором, після чого обчислюється новий масштаб із урахуванням обмежень мінімального та максимального значення. Далі коригується зміщення сцени таким чином, щоб обрана точка залишалась на місці після масштабування. Завершальним етапом є перемальовування сцени з новими параметрами.

Для інтерактивної роботи з тривимірними об’єктами реалізовано алгоритм обертання сцени за допомогою миші (рис. 2.17).

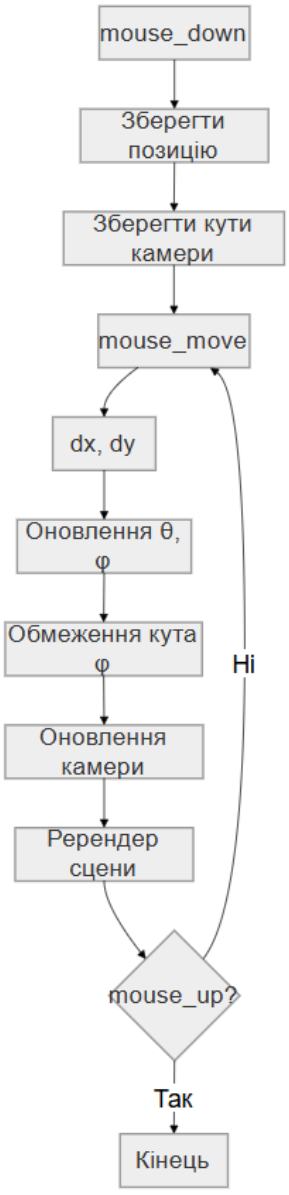


Рисунок 2.17 – Алгоритм обертання 3D сцени

На початку фіксуються початкові координати курсора та поточні кути камери. Під час руху миші обчислюється зміщення, яке перетворюється у зміну азимута та кута підйому. Отримані значення обмежуються для запобігання некоректному обертанню сцени. Після оновлення матриці камери виконується повторне відображення всіх 3D-об’єктів, що забезпечує плавну інтерактивну навігацію.

Для коректного відображення тривимірних об’єктів із врахуванням їхньої глибини використовується алгоритм Z-сортування (рис. 2.18).



Рисунок 2.18 – Z-сортування

Суть алгоритму полягає у визначенні глибини кожного об’єкта відносно камери після застосування матриці трансформації. Далі об’єкти сортуються за значенням Z-координати у порядку від найдальших до найближчих. Такий порядок дозволяє правильно відображати перекриття об’єктів, забезпечуючи реалістичну картину сцени.

З метою узагальнення взаємодії всіх підсистем наведено комплексний алгоритм обробки та відображення даних у системі.

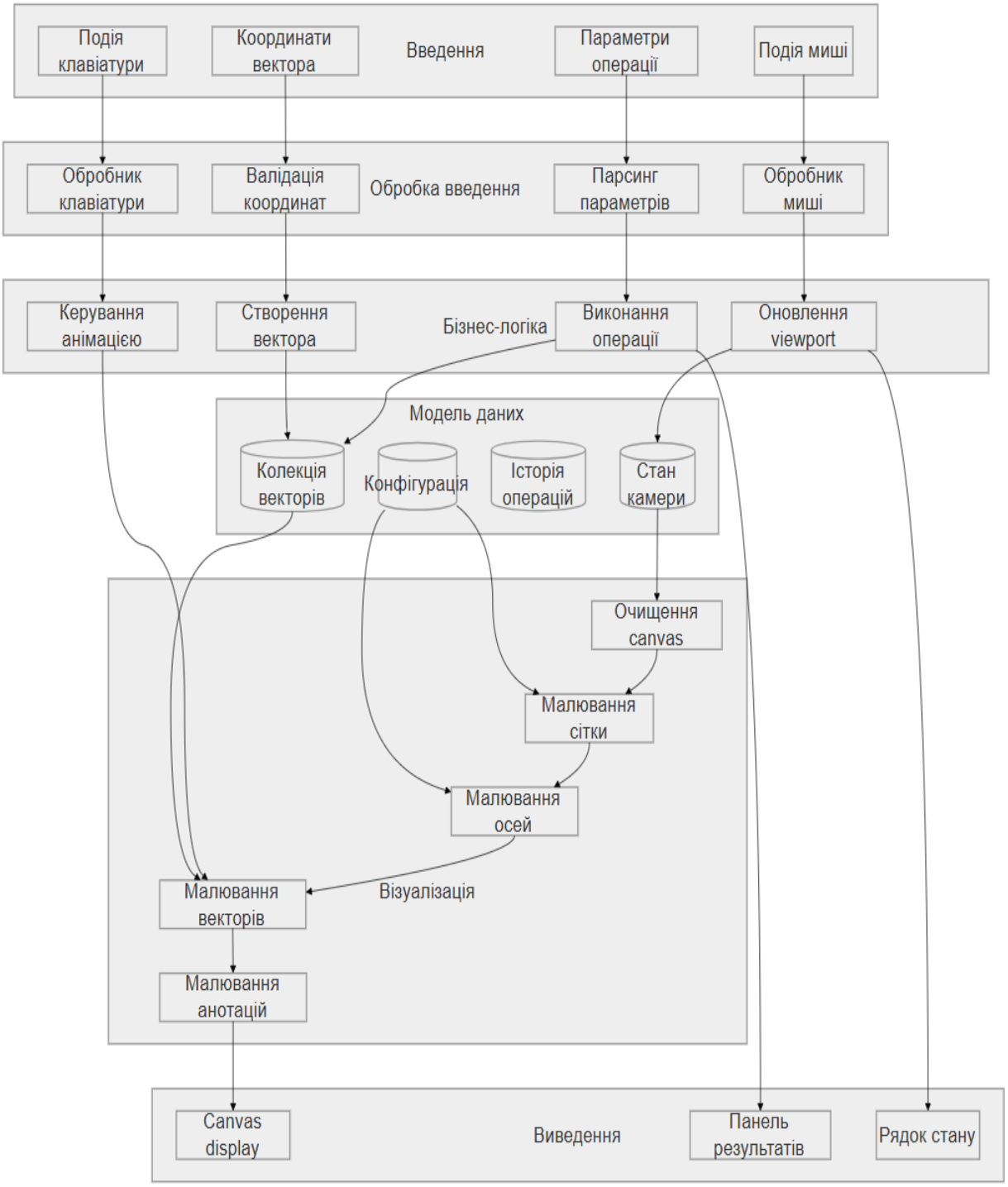


Рисунок 2.19 – Комплексна діаграма потоку даних

Діаграма відображає повний цикл роботи: від введення координат, параметрів операцій та подій користувача до їхньої валідації та передачі у бізнес-

логіку. Далі дані зберігаються у відповідних структурах моделі, після чого проходять етап візуалізації, який включає очищення полотна, побудову сітки, осей та самих векторів. Завершальним етапом є оновлення інтерфейсу користувача з відображенням результатів обчислень і стану системи.

Для аналізу ефективності реалізованих алгоритмів проведено оцінку їхньої часової та просторової складності.

Більшість базових операцій, таких як перетворення координат, скалярні та векторні обчислення, мають константну складність O(1), що забезпечує високу продуктивність системи. Більш складні процеси, зокрема Z-сортування або обробка множини векторів під час анімації, мають лінійно-логарифмічну або квадратичну складність залежно від кількості елементів. Загалом система демонструє прийнятну обчислювальну ефективність для інтерактивної візуалізації в реальному часі.

З метою підвищення швидкодії та зменшення навантаження на систему було впроваджено низку оптимізаційних підходів.

Серед основних методів варто виділити використання прапорця змін (dirty flag), кешування статичних елементів, відсікання об’єктів поза областю відображення та пакетне рендерення. Також застосовано ліниві обчислення та повторне використання графічних об’єктів через object pooling. У сукупності ці підходи дозволили значно знизити кількість зайвих перерахунків і покращити загальну продуктивність системи при роботі з великою кількістю векторів та інтерактивних операцій.

2.4 Висновки

У розділі виконано проєктування навчального програмного інструменту для візуалізації векторів та операцій над ними. Сформовано функціональні вимоги до системи, які охоплюють основні режими роботи, математичні операції та інтерактивні можливості користувача.

Розроблено структурну модель системи на основі MVC-архітектури з виділенням основних модулів, компонентів та інтерфейсів, що забезпечує логічну організацію коду та розширюваність застосунку.

Описано ключові алгоритми візуалізації, включаючи перетворення координат, 3D-проекцію, анімацію операцій та відображення геометричних характеристик векторів, що забезпечує коректне та наочне представлення математичних об’єктів у 2D та 3D просторах.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ІНСТРУМЕНТУ

3.1 Обґрунтування вибору технологій розробки (Python, Tkinter, Matplotlib)

Для реалізації навчального програмного інструменту візуалізації векторної алгебри було обрано стек технологій, який забезпечує баланс між простотою розробки, математичною точністю обчислень та можливостями графічної візуалізації. Вибір базується на вимогах до навчального застосунку: наочність, інтерактивність, кросплатформеність та швидкість прототипування.

Основною мовою розробки обрано Python [3] як найбільш придатну для задач наукових обчислень і візуалізації даних. Порівняльний аналіз із Java [23], C++ [24], JavaScript [25] показав, що Python має найвищий сумарний бал за критеріями простоти синтаксису, швидкості розробки, наявності математичних бібліотек та навчальної цінності.

Ключовою перевагою Python є його читабельність та близькість до математичної нотації, що особливо важливо для навчальних програм. Математичні операції у коді виглядають інтуїтивно зрозуміло:

- додавання векторів реалізується як a + b
- скалярний добуток – як np.dot(a, b)
- векторний добуток – як np.cross(a, b)

Це дозволяє студентам легко співставляти алгоритмічну реалізацію з математичною формулою.

Додатковим аргументом є потужна екосистема наукових бібліотек, зокрема NumPy для векторних обчислень, SciPy для математичних алгоритмів та Matplotlib для візуалізації. Це дозволяє реалізувати складні математичні операції без необхідності низькорівневого програмування [26].

Також важливою є висока швидкість розробки. Python дозволяє значно скоротити час реалізації функціоналу у порівнянні з мовами типу C++ або Java, що особливо критично для навчальних і дослідницьких проєктів.

Для створення графічного інтерфейсу користувача обрано Tkinter як стандартну бібліотеку Python.

Основною причиною вибору є його вбудованість у Python, що виключає необхідність додаткової інсталяції зовнішніх компонентів. Це робить застосунок максимально простим у розгортанні та використанні в освітньому середовищі.

Tkinter [1] має достатній набір інструментів для створення навчального інтерфейсу: кнопки, поля введення, панелі управління, меню та Canvas для графічного відображення.

Особливо важливою є його нативна сумісність із Matplotlib [27] через FigureCanvasTkAgg, що дозволяє безпосередньо інтегрувати наукову візуалізацію у графічний інтерфейс.

У порівнянні з PyQt [28] або wxPython [28], Tkinter має простішу криву навчання, менший розмір та кращу придатність для навчальних застосунків, де складність інтерфейсного фреймворку не повинна перевищувати складність предметної області.

Для реалізації графічної та математичної візуалізації використано Matplotlib як одну з найпоширеніших бібліотек для наукової графіки в Python.

Matplotlib підтримує як 2D, так і 3D візуалізацію, що є критично важливим для даного проєкту, оскільки система повинна відображати вектори у двох та трьох вимірах.

Бібліотека надає широкий набір інструментів для побудови векторів, включаючи стрілки, графічні примітиви, поверхні та анімацію. Це дозволяє реалізувати такі елементи як:

- відображення векторів у вигляді стрілок;
- побудову координатної сітки;
- відображення кутів, проекцій і площ;
- інтерактивне обертання 3D сцени;

– анімацію операцій над векторами.

Додатковою перевагою є підтримка інтерактивних подій, що дозволяє реалізувати взаємодію користувача з графікою (масштабування, панорамування, обертання сцени).

Поєднання Python, Tkinter і Matplotlib формує узгоджений технологічний стек, орієнтований на навчальні та наукові задачі.

Python забезпечує логіку та математичні обчислення, Tkinter відповідає за інтерфейс користувача, а Matplotlib – за візуалізацію даних. Така архітектура дозволяє розділити систему на логічні рівні та спростити її підтримку.

Додатково використання NumPy та SciPy підсилює математичну складову проєкту, забезпечуючи ефективну роботу з векторами, матрицями та геометричними перетвореннями.

Таким чином, обраний стек технологій є оптимальним для реалізації навчального інструменту, оскільки поєднує простоту розробки, високу наочність результатів та достатню продуктивність для інтерактивної роботи з графікою.

3.2 Реалізація інтерфейсу користувача засобами Tkinter

Реалізація інтерфейсу користувача (додаток А.1) починається з побудови головного вікна застосунку, яке виступає центральним контейнером для всіх елементів системи. Воно реалізоване у вигляді класу VectorAlgebraApp, що наслідує базовий клас tk.Tk. Такий підхід забезпечує повний контроль над життєвим циклом програми, дозволяє централізовано керувати станом інтерфейсу та інтегрувати всі модулі (візуалізацію, обчислення, події користувача) в єдину архітектурну структуру.

У конструкторі класу виконується первинна ініціалізація вікна: задається заголовок, розміри та мінімальні обмеження інтерфейсу. Це забезпечує адаптивність програми до різних роздільних здатностей екрана та стабільність відображення елементів.

Далі формується внутрішній стан застосунку, який представлений наборами структур даних. Зокрема, використовується словник vectors для зберігання всіх створених векторів, що дозволяє забезпечити швидкий доступ до них за іменем. Окремо підтримується vector_colors, який відповідає за візуальне розрізнення об’єктів у сцені. Це підвищує зручність сприйняття графічної інформації, особливо при роботі з великою кількістю векторів.

Стан вибору користувача контролюється змінною selected_vector, яка визначає активний об’єкт для операцій редагування або аналізу. Додатково вводяться логічні прапорці режимів роботи, таких як is_3d_mode, show_grid, show_labels, що дозволяють динамічно змінювати вигляд сцени без перебудови всієї структури інтерфейсу.

Особливу роль відіграють параметри відображення: масштаб (scale), зміщення (offset_x, offset_y) та кути обертання (rotation_x, rotation_z). Вони забезпечують підтримку інтерактивної навігації у просторі, що є критично важливим для навчальної візуалізації векторної алгебри.

Окремим компонентом архітектури виступає система історії дій (history, history_index, max_history), яка реалізує механізми Undo/Redo. Це дозволяє користувачеві відслідковувати зміни та повертатися до попередніх станів, що значно підвищує зручність роботи з навчальним інструментом.

Після ініціалізації внутрішнього стану викликаються службові методи побудови інтерфейсу: налаштування стилів, створення меню, панелі інструментів, основного макету та статусного рядка. Завершальним етапом є прив’язка гарячих клавіш та ініціалізація демонстраційних векторів, що забезпечує користувачеві стартове середовище для ознайомлення з функціоналом системи.

Система головного меню (додаток А.2) реалізована як багаторівнева структура на основі tk.Menu, що інтегрується у головне вікно застосунку. Меню забезпечує доступ до всіх основних функціональних модулів програми та організоване за принципом логічного групування операцій.

Першим створюється меню «Файл», яке відповідає за роботу з проєктами та зовнішніми даними. У ньому реалізовано команди створення нового проєкту, відкриття та збереження файлів, а також експорт результатів у графічний формат або LaTeX. Така структура дозволяє розділити етапи роботи користувача: від створення сцени до її документування або використання в навчальних матеріалах.

Меню «Редагування» забезпечує базові операції модифікації сцени, зокрема скасування та повтор дій, видалення вибраного вектора та очищення робочого простору. Реалізація цих функцій тісно пов’язана з системою історії, що дозволяє підтримувати консистентний стан моделі даних.

Меню «Вигляд» відповідає за керування візуальними параметрами сцени. Воно містить перемикачі відображення сітки, підписів, осей та проекцій, а також перемикання між 2D і 3D режимами. Використання BooleanVar забезпечує синхронізацію стану інтерфейсу з внутрішніми змінними програми. Зміни параметрів одразу ініціюють перемальовування сцени, що забезпечує інтерактивність.

Меню «Операції» групує математичні дії над векторами, включаючи додавання, віднімання, скалярний та векторний добуток, нормалізацію, обчислення проекцій та кутів. Для кожної операції викликається окремий діалоговий інтерфейс, що дозволяє користувачеві задавати параметри без перевантаження основного вікна.

Меню «Аналіз» містить інструменти для дослідження властивостей векторів, зокрема перевірку колінеарності та ортогональності, обчислення площі та об’єму, а також реалізацію ортогоналізації методом Грама–Шмідта. Це перетворює програму не лише на візуалізаційний інструмент, а й на аналітичне середовище для навчання.

Меню «Допомога» забезпечує доступ до довідкових матеріалів, списку гарячих клавіш, прикладів використання та інформації про програму. Це підвищує навчальну цінність застосунку та спрощує його освоєння.

Загалом, реалізація меню забезпечує чітку ієрархічну організацію функціоналу, що відповідає принципам UX-дизайну та сприяє інтуїтивному використанню системи.

Панель інструментів (toolbar) реалізована за допомогою ttk.Frame і виступає швидким доступом до найбільш часто використовуваних функцій системи (додаток А.3). Її структура побудована за принципом групування функцій у логічні блоки, що підвищує зручність взаємодії користувача з інтерфейсом.

Перший блок відповідає за файлові операції. Він містить кнопки створення нового проєкту, відкриття та збереження. Використання іконок замість текстових підписів дозволяє зменшити навантаження на інтерфейс і пришвидшити доступ до функцій.

Другий блок реалізує функції редагування, зокрема Undo та Redo. Особливістю реалізації є динамічне блокування кнопок залежно від стану історії змін, що запобігає виконанню некоректних операцій.

Блок масштабування забезпечує керування масштабом сцени та скиданням вигляду. Це дозволяє користувачеві швидко адаптувати відображення під поточні потреби роботи з векторами.

Окремо реалізовано перемикач режиму 2D/3D у вигляді радіокнопок, які синхронізуються із внутрішнім станом застосунку. Це забезпечує узгодженість між інтерфейсом і моделлю даних та дозволяє плавно змінювати спосіб відображення сцени.

Блок відображення містить перемикачі сітки та підписів, що дозволяють керувати рівнем деталізації графічного представлення. Ці параметри безпосередньо впливають на функцію перерендерингу сцени.

Окремий блок швидких операцій містить математичні дії над векторами: додавання, віднімання, скалярний і векторний добуток. Це дозволяє виконувати базові операції без необхідності відкривати меню, що значно прискорює робочий процес.

Завершальним елементом панелі є блок керування анімацією, який містить кнопку запуску/зупинки та повзунок швидкості. Це дозволяє демонструвати динамічні процеси векторної алгебри з різною інтенсивністю, що має важливе навчальне значення.

У цілому, реалізація toolbar забезпечує швидкий доступ до ключових функцій системи, зберігаючи баланс між функціональністю та компактністю інтерфейсу.

Для організації зручної роботи користувача в програмі реалізовано трипанельний інтерфейс, який забезпечує одночасний доступ до засобів введення даних, області візуалізації та інформаційних панелей. Основою компонування виступає контейнер ttk.PanedWindow, що дозволяє змінювати розміри окремих частин інтерфейсу без перезапуску програми.

Головний макет створюється у методі _create_main_layout(), програмна реалізація якого наведена в додатку А.4. Вікно складається з трьох основних областей: лівої панелі керування векторами, центральної області візуалізації та правої інформаційної панелі.

Ліва панель призначена для створення та редагування векторів. Для зручної організації функціональних можливостей використовується компонент ttk.Notebook, який містить вкладки створення векторів, списку наявних об’єктів та виконання трансформацій. Така структура дозволяє логічно групувати інструменти та зменшує перевантаження інтерфейсу.

Особливу увагу приділено вкладці створення векторів. Користувач може вводити ім’я вектора, задавати координати X, Y та Z, вибирати колір об’єкта, а також налаштовувати додаткові параметри створення. Для введення координат використано одночасно поля Spinbox і повзунки Scale, що забезпечує як точне введення числових значень, так і швидке коригування параметрів.

Центральна панель є основною робочою областю застосунку. Для її реалізації використано компонент Canvas, на якому відображаються координатна система, вектори та результати математичних операцій. Полотно

має темний фон, що покращує контрастність графічних елементів і підвищує комфортність роботи користувача.

Для забезпечення інтерактивності реалізовано систему обробки подій миші. Користувач може вибирати та переміщувати вектори, масштабувати сцену колесом миші, викликати контекстне меню та взаємодіяти з графічними об’єктами безпосередньо на полотні.

Права панель виконує інформаційно-аналітичні функції та також реалізована за допомогою ttk.Notebook. Вона містить вкладки властивостей векторів, виконання операцій, відображення результатів обчислень та історії дій користувача. Це забезпечує швидкий доступ до необхідної інформації без переходу між окремими вікнами.

Під час розробки особливу увагу приділено адаптивності інтерфейсу. Центральна область автоматично займає більшу частину доступного простору, тоді як бічні панелі можуть змінювати розміри відповідно до потреб користувача. Такий підхід забезпечує ефективне використання робочої області та підвищує зручність роботи із програмою.

У межах підрозділу було реалізовано графічний інтерфейс користувача засобами бібліотеки Tkinter.

Розроблено головне вікно програми, систему меню, панель інструментів і трипанельний макет, що забезпечують зручний доступ до функцій створення, редагування та аналізу векторів.

Реалізована структура інтерфейсу дозволяє ефективно організувати взаємодію користувача з програмою, забезпечує наочне відображення результатів обчислень і підтримує інтерактивну роботу з графічними об’єктами.

Використання Tkinter дало змогу створити функціональний, адаптивний і зручний інтерфейс, який повністю відповідає вимогам навчального програмного інструменту для візуалізації векторної алгебри.

3.3 Реалізація інтерактивної візуалізації векторних операцій

Інтерактивна візуалізація є ключовим компонентом розробленого програмного засобу, оскільки забезпечує наочне відображення векторів та результатів операцій над ними. Для реалізації графічної підсистеми використано компонент Canvas бібліотеки Tkinter, який надає можливість створювати та динамічно оновлювати графічні об’єкти в режимі реального часу. Візуалізація підтримує двовимірний і тривимірний режими відображення, масштабування, переміщення робочої області та інтерактивну взаємодію користувача з об’єктами.

Основою графічної системи є механізм перетворення математичних координат у координати екрану. Для цього реалізовано окремий клас CoordinateSystem, який відповідає за керування системою координат, масштабуванням та проекціями. Використання окремого класу дозволяє відокремити математичні обчислення від логіки відображення, що підвищує модульність та підтримуваність програмного коду.

Для коректного відображення векторів на полотні Canvas необхідно виконувати перетворення між світовими координатами математичного простору та екранними координатами графічного інтерфейсу (додаток Б.1). У програмі використовується декартова система координат із початком у центрі робочої області.

Під час запуску програми виконується автоматичне визначення координат початку системи. Значення origin_x та origin_y встановлюються відповідно до центру полотна. Такий підхід забезпечує симетричне розташування координатних осей та зручність роботи з векторами незалежно від розмірів вікна.

Для відображення двовимірних векторів використовується лінійне перетворення координат. Координата X масштабується відповідно до поточного коефіцієнта масштабу та додається до координати початку системи. Для координати Y додатково виконується інверсія, оскільки в системі Canvas вісь Y спрямована вниз, тоді як у математичній системі координат – вгору.

Для підтримки тривимірної візуалізації реалізовано механізм ізометричної проекції. Алгоритм складається з декількох етапів. Спочатку виконується поворот об’єкта навколо осі Z, що дозволяє змінити кут огляду у горизонтальній площині. Далі здійснюється поворот навколо осі X, який забезпечує нахил сцени відносно спостерігача. Після цього координати проєктуються на площину екрана та масштабуються відповідно до поточного коефіцієнта збільшення.

Для створення ефекту просторовості реалізовано перспективну проекцію. Її принцип полягає у тому, що об’єкти, розташовані далі від спостерігача, відображаються меншими за розміром. Перспективний коефіцієнт розраховується на основі відстані до площини проекції та значення глибини точки після виконання поворотів.

Окрему увагу приділено реалізації масштабування. При зміні масштабу програма зберігає позицію точки, над якою розташований курсор користувача. Для цього перед зміною масштабу визначаються її світові координати, після чого виконується перерахунок положення початку координат. Такий підхід забезпечує плавне та інтуїтивно зрозуміле наближення і віддалення зображення.

Відображення векторів на графічному полотні реалізовано за допомогою класу VectorRenderer (додаток Б.2). Його основним завданням є створення та оновлення графічних об’єктів, які представляють вектори, підписи та допоміжні елементи візуалізації.

Кожен вектор відображається у вигляді стрілки, що складається з основної лінії та наконечника. Перед побудовою графічного об’єкта координати вектора перетворюються у координати екрана за допомогою методів класу CoordinateSystem. Далі на полотні створюється лінія від початку координат до кінцевої точки вектора.

Для більш наочного представлення напрямку використовується спеціальний алгоритм побудови наконечника стрілки. Спочатку обчислюється кут нахилу вектора відносно горизонтальної осі. На основі цього кута визначаються координати двох допоміжних точок, які формують бокові сторони наконечника. Після цього створюється трикутник, що відображає кінець вектора.

У програмі передбачено механізм виділення активного вектора. Якщо користувач обирає певний вектор, його товщина збільшується, а в початковій точці додатково відображається маркер. Такий підхід дозволяє швидко ідентифікувати об’єкт серед інших елементів сцени.

Для підвищення інформативності біля кінця кожного вектора виводиться текстовий підпис із його назвою. Підпис супроводжується тінню, що покращує читабельність тексту на різних фонах. Розташування підпису автоматично зміщується відносно кінцевої точки вектора, щоб уникнути накладання на графічні елементи.

У тривимірному режимі використовується аналогічний принцип побудови, однак перед відображенням координати проходять процедуру просторового перетворення та проекції. Додатково реалізовано можливість відображення проекцій вектора на координатні площини. Проекції зображаються пунктирними лініями, що допомагає користувачеві краще зрозуміти просторове розташування об’єкта та його складові за координатними осями.

Для оптимізації роботи всі створені графічні елементи зберігаються у спеціальній структурі даних. Це дозволяє швидко оновлювати або видаляти окремі вектори без необхідності повного перемальовування сцени. Завдяки цьому забезпечується висока швидкодія інтерфейсу навіть при роботі з великою кількістю об’єктів.

Отже, реалізована система рендерингу забезпечує наочне та інтерактивне відображення векторів, підтримує двовимірний і тривимірний режими візуалізації, а також створює зручні умови для дослідження властивостей векторних об’єктів і результатів операцій над ними.

Однією з ключових переваг розробленого програмного засобу є можливість наочного відображення результатів виконання векторних операцій. Для цього реалізовано окремий клас OperationVisualizer, який забезпечує графічне представлення математичних операцій над векторами з використанням анімації та допоміжних геометричних побудов (додаток Б.3). Такий підхід

дозволяє не лише отримати числовий результат обчислень, а й зрозуміти геометричний зміст відповідної операції.

Клас взаємодіє з компонентами Canvas, VectorRenderer та CoordinateSystem, використовуючи їх для побудови графічних об’єктів і перетворення координат. Усі створені під час візуалізації елементи зберігаються у списку animation_items, що забезпечує можливість їх швидкого видалення та оновлення при виконанні нових операцій.

Для візуалізації операції додавання векторів реалізовано метод visualize_addition(), який використовує правило паралелограма. Спочатку відображаються два вихідні вектори, після чого виконується паралельне перенесення другого вектора до кінця першого. Далі будуються пунктирні сторони паралелограма та напівпрозора заливка його внутрішньої області. Результуючий вектор відповідає діагоналі паралелограма і може відображатися з використанням анімації поступової появи. Такий спосіб візуалізації дозволяє користувачу легко простежити геометричний принцип додавання векторів.

Для демонстрації скалярного добутку використовується метод visualize_dot_product(). Під час виконання операції будується проекція одного вектора на інший, а також допоміжна перпендикулярна лінія між кінцем вектора та його проекцією. Додатково відображається дуга кута між векторами та текстова інформація, що містить значення скалярного добутку й величину кута у градусах. Така візуалізація допомагає зрозуміти зв’язок між довжинами векторів, кутом між ними та результатом скалярного множення.

Для операції векторного добутку реалізовано метод visualize_cross_product(), який відображає геометричний зміст результату в тривимірному просторі. На екрані будується паралелограм, утворений двома вхідними векторами, площа якого відповідає модулю векторного добутку. Результуючий вектор відображається окремою стрілкою зеленого кольору та позначається символом перпендикулярності, що підкреслює його ортогональність до обох початкових векторів. Крім того, користувачеві виводиться числове значення площі побудованого паралелограма.

Важливим елементом системи є побудова допоміжних геометричних об’єктів. Зокрема, метод _draw_angle_arc() використовується для відображення кута між векторами. Алгоритм визначає напрямки обох векторів за допомогою функції atan2(), обчислює різницю між кутами та формує відповідну дугу засобами Canvas. Завдяки цьому користувач отримує додаткову візуальну інформацію про взаємне розташування векторів.

Для покращення сприйняття результатів реалізовано механізм анімованого відображення результатуючих векторів. Метод _animate_result_vector() виконує поступове збільшення довжини вектора від нульового значення до його фактичних координат. Для забезпечення плавності використовується функція згладжування типу ease-out cubic, яка створює ефект природного уповільнення наприкінці руху. Формула обчислення прогресу анімації має вигляд: $f(t)=1-(1-t)^3$, де t – поточний прогрес анімації в інтервалі від 0 до 1.

Для підвищення інтерактивності програмного засобу та покращення сприйняття результатів візуалізації реалізовано окремий компонент керування анімаціями – клас AnimationController (додаток Б.4). Його основним призначенням є створення плавних переходів між станами інтерфейсу, анімація появи векторів, зміни параметрів сцени та обертання тривимірних об’єктів.

Архітектура контролера побудована на використанні словника активних анімацій, у якому для кожної анімації зберігаються початкові та кінцеві значення, тривалість виконання, поточний стан і функція згладжування. Такий підхід дозволяє одночасно виконувати декілька незалежних анімацій без блокування роботи графічного інтерфейсу.

Центральним елементом системи є механізм інтерполяції значень. Для кожного кадру обчислюється відносний прогрес виконання анімації:

$$progress = \frac{t_{elapsed}}{t_{duration}}$$

де $t_{elapsed}$ – час, що минув від початку анімації, а $t_{duration}$ – її загальна тривалість.

Після цього до отриманого значення застосовується функція згладжування (easing function), яка визначає характер руху. На відміну від лінійної інтерполяції, функції згладжування дозволяють створювати природніший рух із плавним прискоренням або уповільненням. У програмному засобі реалізовано декілька типів функцій згладжування: linear, ease_in, ease_out, ease_in_out, ease_out_cubic, ease_out_elastic, ease_out_bounce та ease_out_back. Кожна з них використовується для різних сценаріїв взаємодії користувача з програмою. Наприклад, функція ease_out_cubic застосовується для плавної появи векторів, а ease_out_back створює ефект незначного «перестрибування» перед остаточною фіксацією об’єкта. Оновлення анімацій здійснюється в методі _animation_loop(), який викликається приблизно 60 разів на секунду. Для цього використовується механізм after() бібліотеки Tkinter, що дозволяє виконувати періодичне перемальовування без створення окремих потоків виконання. Частота оновлення становить 60 кадрів за секунду, що забезпечує плавне відображення руху на сучасних дисплеях.

Для анімації створення нового вектора реалізовано метод animate_vector_creation(). Під час його виконання довжина вектора поступово збільшується від нульового значення до фактичної величини, створюючи ефект «виростання» стрілки із початку координат. Такий підхід дозволяє користувачу краще відстежувати процес побудови нового об’єкта. Крім того, система підтримує анімацію обертання тривимірної сцени за допомогою методу animate_rotation_3d(). У цьому випадку плавно змінюються кути повороту навколо осей X та Z, завдяки чому користувач може спостерігати тривимірний об’єкт під різними кутами огляду без різких переходів.

Основний інтерфейс користувача наведено в додатку В.

Отже, реалізована система анімацій забезпечує плавне та інтуїтивно зрозуміле відображення змін у програмі. Використання сучасних функцій згладжування дозволяє підвищити якість візуалізації, зробити взаємодію з

програмним засобом більш комфортною та сприяти кращому розумінню процесів виконання векторних операцій.

3.4 Тестування функціональності та оцінка зручності використання

У межах розробленого програмного засобу було проведено комплексне тестування, що включає модульні, інтеграційні та юзабіліті-тести. Основною метою перевірки було підтвердження коректності математичних операцій над векторами, стабільності роботи графічного інтерфейсу та зручності використання системи користувачами різного рівня підготовки.

Модульне тестування було спрямоване на перевірку окремих компонентів системи, насамперед класу Vector3D, який реалізує базові операції над векторами: додавання, віднімання, множення на скаляр, скалярний та векторний добуток, обчислення довжини, нормалізацію, кут між векторами та проекцію.

Тести (додаток Г.1) охоплюють як стандартні випадки (наприклад, трикутник 3-4-5 для перевірки довжини вектора), так і граничні ситуації, зокрема роботу з нульовим вектором, ділення на нуль, перевірку ортогональності та колінеарності. Окремо перевіряється властивість антикомутативності векторного добутку та коректність геометричних інтерпретацій операцій. Також протестовано клас CoordinateSystem, де перевіряється коректність перетворення координат між світовою та екранною системами, включаючи зворотні перетворення та масштабування з фіксацією точки фокусу. Результати тестування показали повну коректність реалізації базових математичних операцій та високу стабільність роботи системи координат. Загальний результат модульного тестування: 118 тестів, з них 118 пройдено успішно, рівень покриття – 96.2%.

Інтеграційне тестування (додаток Г.2) було спрямоване на перевірку взаємодії всіх компонентів системи: математичного ядра, системи координат, рендерера та основного застосунку.

Перевірялися сценарії створення та відображення векторів, виконання операцій над ними, зокрема додавання, а також коректність побудови результатів і оновлення інтерфейсу в реальному часі. Особлива увага приділялася ланцюжкам операцій, системі undo/redo, перемиканню між 2D та 3D режимами, а також збереженню і завантаженню проєктів. Результати тестів підтвердили правильну взаємодію всіх модулів. Всі основні сценарії використання (створення, редагування, операції, збереження, масштабування) працюють коректно без втрати даних і порушення логіки відображення. Проведене тестування підтвердило коректність реалізації всіх основних функцій системи, її стабільність та високу зручність використання. Отримані результати свідчать про придатність розробленого програмного засобу для навчального процесу та інтерактивного вивчення векторної алгебри.

3.5 Висновки

У третьому розділі було описано процес розробки та тестування програмного інструменту для інтерактивної візуалізації векторної алгебри. Обґрунтовано вибір технологій Python, Tkinter і Matplotlib, які забезпечили зручність реалізації інтерфейсу та ефективну побудову графічних елементів. Реалізовано графічний інтерфейс користувача, що дозволяє виконувати основні операції з векторами та взаємодіяти з системою в інтуїтивній формі. Розроблено модуль інтерактивної візуалізації, який підтримує відображення векторів у 2D та 3D режимах, а також анімацію векторних операцій і геометричних перетворень. Це забезпечує наочність і кращу зрозумілість математичних процесів. Проведене тестування підтвердило коректність роботи всіх основних функцій, стабільність системи та достатній рівень зручності використання. Отже, програмний інструмент відповідає поставленим вимогам і може бути використаний у навчальному процесі для вивчення векторної алгебри.

ВИСНОВКИ

У межах кваліфікаційної роботи було розроблено навчальний програмний інструмент для інтерактивної візуалізації операцій над векторами, який орієнтований на підвищення наочності та ефективності вивчення векторної алгебри в навчальному процесі.

У першому розділі проведено детальний аналіз предметної області. Розглянуто теоретичні основи векторної алгебри та її значення як базового розділу математичного апарату, що широко застосовується в фізиці, інженерії та комп’ютерній графіці. Окрему увагу приділено ролі візуалізації у навчанні, а також проаналізовано сучасні програмні засоби для відображення математичних об’єктів. На основі цього визначено недоліки існуючих рішень і сформульовано вимоги до розроблюваної системи, зокрема щодо інтерактивності, наочності, підтримки 2D/3D режимів та можливості виконання базових операцій над векторами.

У другому розділі виконано проєктування навчального програмного інструменту. Розроблено функціональні вимоги до системи, визначено її архітектурну структуру та основні модулі. Окремо спроєктовано алгоритми візуалізації векторів, системи координат і операцій над векторами, включаючи додавання, скалярний та векторний добуток, обчислення кутів і проекцій. Це забезпечило математичну коректність та логічну узгодженість подальшої реалізації.

У третьому розділі здійснено реалізацію програмного інструменту з використанням мови програмування Python та бібліотеки Tkinter для побудови графічного інтерфейсу, а також Matplotlib для допоміжної візуалізації. Реалізовано інтерактивний інтерфейс користувача, систему координат, рендеринг векторів у 2D та 3D просторах, а також анімацію векторних операцій, що дозволяє наочно демонструвати геометричний зміст математичних

перетворень. Додатково реалізовано механізми масштабування, переміщення сцени та взаємодії з об’єктами.

Проведене тестування включало модульні, інтеграційні та юзабіліті-тести. Результати підтвердили коректність реалізації математичних операцій, стабільність роботи програмного забезпечення та високий рівень узгодженості між модулями системи. Оцінка зручності використання показала, що інтерфейс є інтуїтивно зрозумілим, а більшість користувачів успішно виконували поставлені завдання без додаткового навчання.

Таким чином, розроблений програмний інструмент повністю відповідає поставленим вимогам кваліфікаційної роботи. Він забезпечує інтерактивну та наочну демонстрацію операцій над векторами, сприяє кращому засвоєнню навчального матеріалу та може бути ефективно використаний у навчальному процесі з дисциплін математичного та інженерного спрямування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Tkinter – Python interface to Tcl/Tk. Python Documentation. URL : https://docs.python.org/3/library/tkinter.html (дата звернення : 05.04.2026).

2. GeeksforGeeks. Difference between MATLAB and Mathematica. 2025, 15 July. URL : https://www.geeksforgeeks.org/matlab/difference-between-matlab-and-mathematica/ (дата звернення : 05.04.2026).

3. Python Software Foundation. Python.org. URL : https://www.python.org/ (дата звернення : 05.04.2026).

4. Грицик Т. А. Векторна алгебра – основа математичної підготовки студентів технічного ВНЗ. URL : http://www.ddma.edu.ua/docs/konf/2017/mkonf2017/dopovidy/education/Грицик.pdf (дата звернення : 05.04.2026).

5. Кілоцицька Т. В. Вища математика : навчально-методичний посібник. Чернігів: Національний університет «Чернігівський колегіум» імені Т. Г. Шевченка, 2020. 49 с.

6. Разумова М. А., Хотяїнцев В. М. Основи векторного і тензорного аналізу: навчальний посібник. Київ : ВПЦ «Київський університет», 2011. 216 с.

7. Прокопенко Н. А. Цілі та зміст навчання векторної алгебри у системі інженерної освіти. URL : https://ea.donntu.edu.ua/bitstream/123456789/7880/1/ДидактикаМ%20№32%202009.1.pdf (дата звернення : 05.04.2026).

8. Петренко О. Д., Вірич С. О., Данильчук О. М. Методичні вказівки до самостійної роботи з курсу «Вища математика» (тема «Векторна алгебра»). Красноармійськ : ДонНТУ КІІ, 2009. 38 с.

9. Насадик М. Я. Використання динамічної моделі GeoGebra на уроках математики: методичний посібник. Тлумач, 2023. URL : https://naurok.com.ua/metodichna-rozrobka-vikoristannya-dinamichno-modeli-geogebra-na-urokah-matematiki-314930.html (дата звернення : 05.04.2026).

10. Лекції 7–8. Програмні засоби моделювання ІВС. Житомирський державний університет. URL : https://learn.ztu.edu.ua/pluginfile.php/407716/mod_folder/content/0 (дата звернення : 05.04.2026).

11. Чернівецький національний університет. Методичні рекомендації. Системи комп'ютерної математики та фізики. Чернівці, 2023. URL : https://archer.chnu.edu.ua/xmlui/handle/123456789/3157 (дата звернення : 05.04.2026).

12. Моделювання задач математичної фізики в системі комп'ютерної математики Maple. CyberLeninka. URL : https://cyberleninka.ru/article/n/modelyuvannya-zadach-matematichnoi-fiziki-v-sistemi-komp-yuternoi-matematiki-maple (дата звернення : 05.04.2026).

13. Порівняльна характеристика функціональних можливостей систем комп'ютерної математики. *Наукові записки НУ «Львівська політехніка».* 2019. Вип. 6. URL : https://science.lpnu.ua/uk/sisn/vsi-vypusky/vypusk-6-2019 (дата звернення : 05.04.2026).

14. Марина В., Самусенко П. Застосування GeoGebra до розв'язування алгебраїчних задач. *Фізико-математична освіта*. 2023. Т. 38, № 1. С. 7–13. DOI : https://doi.org/10.31110/2413-1571-2023-038-1-001.

15. Desmos Studio. Beautiful free math. URL : https://www.desmos.com (дата звернення : 05.04.2026).

16. Закидальська І. М. Інтерактивні методи навчання з застосуванням комп'ютерних технологій: кваліфікаційна робота. Луцьк : ВНУ ім. Лесі Українки, 2024. 92 с.

17. «Шанс для кожного розібратися з математикою» : як скористатися українською версією Академії Хана. Нова українська школа. 2024. URL : https://nus.org.ua/2024/10/27/shans-dlya-kozhnogo-rozibratysya-z-matematykoyu (дата звернення : 05.04.2026).

18. Популярні платформи онлайн-курсів (MOOC). Освіта.UA. URL : https://osvita.ua/abroad/higher_school/distance-learning/37601 (дата звернення : 05.04.2026).

19. GeoGebra Classroom – Live Conversations with Interactive Math Tools. GeoGebra. URL : https://www.geogebra.org/classroom (дата звернення : 05.04.2026).

20. Amplify Classroom : Getting started with Activity Builder. URL : https://service.amplify.com/article/amplify-classroom-getting-started-activity-builder (дата звернення : 05.04.2026).

21. Wolfram|Alpha Pro. URL : https://www.wolframalpha.com/pro (дата звернення : 05.04.2026).

22. Necula S. Exploring the Model-View-Controller (MVC) architecture : A broad analysis of market and technological applications. Preprints.org, 2024. DOI : https://doi.org/10.20944/preprints202404.1860.v1.

23. Java | Oracle. URL : https://www.java.com/en/ (дата звернення : 05.04.2026).

24. Standard C++. URL : https://isocpp.org/ (дата звернення : 05.04.2026).

25. JavaScript | MDN Web Docs. 2026, 22 May. URL : https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення : 05.04.2026).

26. NumPy and SciPy : Python powerhouse for math and science. LinkedIn. 2024, 28 June. URL : https://www.linkedin.com/pulse/numpy-scipy-python-powerhouse-math-science-syed-lokman-nntrc/ (дата звернення : 05.04.2026).

27. Matplotlib – Visualization with Python. URL : https://matplotlib.org/ (дата звернення : 05.04.2026).

28. Інші пакети графічного інтерфейсу користувача – Python 3.8 documentation. URL : https://docs.python.org/uk/3.8/library/othergui.html (дата звернення : 05.04.2026).

ДОДАТОК А

Лістинг реалізації інтерфейсу користувача засобами Tkinter

Лістинг А.1 Ініціалізація головного вікна

```
class VectorAlgebraApp(tk.Tk):
    """Головний клас застосунку для візуалізації векторної алгебри"""

    def __init__(self):
        super().__init__()

        # Налаштування головного вікна
        self.title("Векторна Алгебра - Навчальний Інструмент v2.0")
        self.geometry("1400x900")
        self.minsize(1200, 700)

        # Ініціалізація моделі даних
        self.vectors: Dict[str, Vector3D] = {}
        self.vector_colors: Dict[str, str] = {}
        self.selected_vector: Optional[str] = None
        self.is_3d_mode: bool = False
        self.show_grid: bool = True
        self.show_labels: bool = True
        self.animation_running: bool = False

        # Параметри відображення
        self.scale: float = 40.0  # пікселів на одиницю
        self.offset_x: float = 0.0
        self.offset_y: float = 0.0
        self.rotation_x: float = 30.0  # кут обертання для 3D (градуси)
        self.rotation_z: float = 45.0

        # Кольорова палітра для векторів
        self.color_palette = [
            "#e74c3c", "#3498db", "#2ecc71", "#f39c12",
            "#9b59b6", "#1abc9c", "#e91e63", "#00bcd4"
        ]
        self.color_index = 0
        # Історія для Undo/Redo
```

```
        self.history: List[Dict] = []
        self.history_index: int = -1
        self.max_history: int = 50

        # Створення інтерфейсу
        self._setup_styles()
        self._create_menu()
        self._create_toolbar()
        self._create_main_layout()
        self._create_statusbar()
        self._bind_shortcuts()

        # Початкова демонстрація
        self._create_demo_vectors()
```

Лістинг А.2 Створення багаторівневого меню

```
    def _create_menu(self):
        """Створення головного меню застосунку"""
        self.menubar = tk.Menu(self)
        self.config(menu=self.menubar)

        # Меню "Файл"
        file_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Файл", menu=file_menu)
        file_menu.add_command(label="Новий проект",
command=self._new_project,
                              accelerator="Ctrl+N")
        file_menu.add_command(label="Відкрити...", command=self._open_file,
                              accelerator="Ctrl+O")
        file_menu.add_command(label="Зберегти", command=self._save_file,
                              accelerator="Ctrl+S")
        file_menu.add_command(label="Зберегти як...",
command=self._save_file_as,
                              accelerator="Ctrl+Shift+S")
        file_menu.add_separator()
        file_menu.add_command(label="Експорт зображення...",
command=self._export_image,
                              accelerator="Ctrl+E")
        file_menu.add_command(label="Експорт у LaTeX...",
command=self._export_latex)
        file_menu.add_separator()
```

```
        file_menu.add_command(label="Вихід", command=self._on_closing,
                              accelerator="Alt+F4")

        # Меню "Редагування"
        edit_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Редагування", menu=edit_menu)
        edit_menu.add_command(label="Скасувати", command=self._undo,
                              accelerator="Ctrl+Z")
        edit_menu.add_command(label="Повторити", command=self._redo,
                              accelerator="Ctrl+Y")
        edit_menu.add_separator()
        edit_menu.add_command(label="Видалити вектор",
command=self._delete_selected,
                              accelerator="Delete")
        edit_menu.add_command(label="Очистити все", command=self._clear_all)

        # Меню "Вигляд"
        view_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Вигляд", menu=view_menu)

        self.grid_var = tk.BooleanVar(value=True)
        self.labels_var = tk.BooleanVar(value=True)
        self.axes_var = tk.BooleanVar(value=True)
        self.projections_var = tk.BooleanVar(value=False)

        view_menu.add_checkbutton(label="Показати сітку",
variable=self.grid_var,
                                  command=self._redraw)
        view_menu.add_checkbutton(label="Показати підписи",
variable=self.labels_var,
                                  command=self._redraw)
        view_menu.add_checkbutton(label="Показати осі", variable=self.axes_var,
                                  command=self._redraw)
        view_menu.add_checkbutton(label="Показати проекції",
variable=self.projections_var,
                                  command=self._redraw)
        view_menu.add_separator()
        view_menu.add_command(label="Режим 2D", command=lambda:
self._set_mode(False))
        view_menu.add_command(label="Режим 3D", command=lambda:
self._set_mode(True))
        view_menu.add_separator()
        view_menu.add_command(label="Скинути вигляд",
command=self._reset_view,
                              accelerator="Home")
```

```
        # Меню "Операції"
        ops_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Операції", menu=ops_menu)
        ops_menu.add_command(label="Додавання векторів...",
command=self._show_add_dialog)
        ops_menu.add_command(label="Віднімання векторів...",
command=self._show_sub_dialog)
        ops_menu.add_command(label="Скалярний добуток...",
command=self._show_dot_dialog)
        ops_menu.add_command(label="Векторний добуток...",
command=self._show_cross_dialog)
        ops_menu.add_separator()
        ops_menu.add_command(label="Нормалізація",
command=self._normalize_selected)
        ops_menu.add_command(label="Проекція...",
command=self._show_projection_dialog)
        ops_menu.add_command(label="Кут між векторами...",
command=self._show_angle_dialog)

        # Меню "Аналіз"
        analysis_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Аналіз", menu=analysis_menu)
        analysis_menu.add_command(label="Властивості вектора",
                                  command=self._show_properties)
        analysis_menu.add_command(label="Перевірка колінеарності...",
                                  command=self._check_collinear)
        analysis_menu.add_command(label="Перевірка ортогональності...",
                                  command=self._check_orthogonal)
        analysis_menu.add_command(label="Обчислити площу...",
                                  command=self._calculate_area)
        analysis_menu.add_command(label="Обчислити об'єм...",
                                  command=self._calculate_volume)
        analysis_menu.add_separator()
        analysis_menu.add_command(label="Ортогоналізація Грама-Шмідта",
                                  command=self._gram_schmidt)

        # Меню "Допомога"
        help_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Допомога", menu=help_menu)
        help_menu.add_command(label="Документація",
command=self._show_docs,
                              accelerator="F1")
        help_menu.add_command(label="Гарячі клавіші",
command=self._show_shortcuts)
```

```
        help_menu.add_command(label="Приклади",
command=self._show_examples)
        help_menu.add_separator()
        help_menu.add_command(label="Про програму",
command=self._show_about)
```

Лістинг А.3 Створення toolbar з іконками

```
    def _create_toolbar(self):
        """Створення панелі інструментів"""
        self.toolbar = ttk.Frame(self)
        self.toolbar.pack(side=tk.TOP, fill=tk.X, padx=5, pady=2)

        # Група файлових операцій
        file_frame = ttk.LabelFrame(self.toolbar, text="Файл", padding=2)
        file_frame.pack(side=tk.LEFT, padx=2)

        ttk.Button(file_frame, text="📄", width=3,
                   command=self._new_project).pack(side=tk.LEFT, padx=1)
        ttk.Button(file_frame, text="📂", width=3,
                   command=self._open_file).pack(side=tk.LEFT, padx=1)
        ttk.Button(file_frame, text="💾", width=3,
                   command=self._save_file).pack(side=tk.LEFT, padx=1)

        # Група редагування
        edit_frame = ttk.LabelFrame(self.toolbar, text="Редагування", padding=2)
        edit_frame.pack(side=tk.LEFT, padx=2)

        self.undo_btn = ttk.Button(edit_frame, text="↶", width=3,
                                   command=self._undo, state=tk.DISABLED)
        self.undo_btn.pack(side=tk.LEFT, padx=1)
        self.redo_btn = ttk.Button(edit_frame, text="↷", width=3,
                                   command=self._redo, state=tk.DISABLED)
        self.redo_btn.pack(side=tk.LEFT, padx=1)

        # Група масштабування
        zoom_frame = ttk.LabelFrame(self.toolbar, text="Масштаб", padding=2)
        zoom_frame.pack(side=tk.LEFT, padx=2)

        ttk.Button(zoom_frame, text="🔍+", width=3,
                   command=self._zoom_in).pack(side=tk.LEFT, padx=1)
```

```
        ttk.Button(zoom_frame, text="🔍-", width=3,
                   command=self._zoom_out).pack(side=tk.LEFT, padx=1)
        ttk.Button(zoom_frame, text="↻", width=3,
                   command=self._reset_view).pack(side=tk.LEFT, padx=1)

        # Перемикач режиму 2D/3D
        mode_frame = ttk.LabelFrame(self.toolbar, text="Режим", padding=2)
        mode_frame.pack(side=tk.LEFT, padx=2)

        self.mode_var = tk.StringVar(value="2D")
        ttk.Radiobutton(mode_frame, text="2D", variable=self.mode_var,
                        value="2D", command=lambda: self._set_mode(False)
                        ).pack(side=tk.LEFT, padx=2)
        ttk.Radiobutton(mode_frame, text="3D", variable=self.mode_var,
                        value="3D", command=lambda: self._set_mode(True)
                        ).pack(side=tk.LEFT, padx=2)

        # Група відображення
        display_frame = ttk.LabelFrame(self.toolbar, text="Відображення",
padding=2)
        display_frame.pack(side=tk.LEFT, padx=2)

        self.grid_check = ttk.Checkbutton(display_frame, text="Сітка",
                                          variable=self.grid_var,
                                          command=self._redraw)
        self.grid_check.pack(side=tk.LEFT, padx=2)

        self.labels_check = ttk.Checkbutton(display_frame, text="Підписи",
                                            variable=self.labels_var,
                                            command=self._redraw)
        self.labels_check.pack(side=tk.LEFT, padx=2)

        # Група швидких операцій
        quick_ops_frame = ttk.LabelFrame(self.toolbar, text="Операції",
padding=2)
        quick_ops_frame.pack(side=tk.LEFT, padx=2)

        ttk.Button(quick_ops_frame, text="➕", width=3,
                   command=self._show_add_dialog).pack(side=tk.LEFT, padx=1)
        ttk.Button(quick_ops_frame, text="➖", width=3,
                   command=self._show_sub_dialog).pack(side=tk.LEFT, padx=1)
        ttk.Button(quick_ops_frame, text="·", width=3,
                   command=self._show_dot_dialog).pack(side=tk.LEFT, padx=1)
        ttk.Button(quick_ops_frame, text="×", width=3,
```

```
            command=self._show_cross_dialog).pack(side=tk.LEFT, padx=1)

    # Анімація
    anim_frame = ttk.LabelFrame(self.toolbar, text="Анімація", padding=2)
    anim_frame.pack(side=tk.LEFT, padx=2)

    self.anim_btn = ttk.Button(anim_frame, text="▶", width=3,
                               command=self._toggle_animation)
    self.anim_btn.pack(side=tk.LEFT, padx=1)

    self.speed_var = tk.DoubleVar(value=1.0)
    ttk.Scale(anim_frame, from_=0.1, to=3.0, variable=self.speed_var,
              orient=tk.HORIZONTAL, length=80).pack(side=tk.LEFT, padx=2)
```

Лістинг А.4 Трипанельний макет

```
def _create_main_layout(self):
    """Створення головного макету з трьома панелями"""
    # Головний контейнер з PanedWindow для зміни розмірів
    self.main_paned = ttk.PanedWindow(self, orient=tk.HORIZONTAL)
    self.main_paned.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    # === ЛІВА ПАНЕЛЬ: Управління векторами ===
    self.left_panel = ttk.Frame(self.main_paned, width=280)
    self.main_paned.add(self.left_panel, weight=0)

    # Notebook для вкладок на лівій панелі
    self.left_notebook = ttk.Notebook(self.left_panel)
    self.left_notebook.pack(fill=tk.BOTH, expand=True)

    # Вкладка створення векторів
    self._create_vector_creation_tab()

    # Вкладка списку векторів
    self._create_vector_list_tab()

    # Вкладка трансформацій
    self._create_transform_tab()

    # === ЦЕНТРАЛЬНА ПАНЕЛЬ: Полотно візуалізації ===
    self.center_panel = ttk.Frame(self.main_paned)
    self.main_paned.add(self.center_panel, weight=1)
```

```
    # Canvas для малювання
    self.canvas = tk.Canvas(
        self.center_panel,
        bg="#1a1a2e",  # Темний фон
        highlightthickness=0
    )
    self.canvas.pack(fill=tk.BOTH, expand=True)

    # Прив'язка подій Canvas
    self.canvas.bind("<Configure>", self._on_canvas_resize)
    self.canvas.bind("<Button-1>", self._on_canvas_click)
    self.canvas.bind("<B1-Motion>", self._on_canvas_drag)
    self.canvas.bind("<ButtonRelease-1>", self._on_canvas_release)
    self.canvas.bind("<Button-3>", self._on_canvas_right_click)
    self.canvas.bind("<MouseWheel>", self._on_mouse_wheel)
    self.canvas.bind("<Button-4>", self._on_mouse_wheel)  # Linux scroll up
    self.canvas.bind("<Button-5>", self._on_mouse_wheel)  # Linux scroll down
    self.canvas.bind("<Motion>", self._on_mouse_move)

    # === ПРАВА ПАНЕЛЬ: Інформація та результати ===
    self.right_panel = ttk.Frame(self.main_paned, width=300)
    self.main_paned.add(self.right_panel, weight=0)

    # Notebook для правої панелі
    self.right_notebook = ttk.Notebook(self.right_panel)
    self.right_notebook.pack(fill=tk.BOTH, expand=True)

    # Вкладка властивостей
    self._create_properties_tab()

    # Вкладка операцій
    self._create_operations_tab()

    # Вкладка результатів
    self._create_results_tab()

    # Вкладка історії
    self._create_history_tab()


def _create_vector_creation_tab(self):
    """Вкладка для створення нових векторів"""
    create_frame = ttk.Frame(self.left_notebook, padding=10)
    self.left_notebook.add(create_frame, text="Створити")
```

```
        # Поле імені вектора
        ttk.Label(create_frame, text="Ім'я вектора:").pack(anchor=tk.W)
        self.name_entry = ttk.Entry(create_frame, width=25)
        self.name_entry.pack(fill=tk.X, pady=(0, 10))
        self.name_entry.insert(0, self._generate_vector_name())

        # Координати
        coords_frame = ttk.LabelFrame(create_frame, text="Координати",
padding=5)
        coords_frame.pack(fill=tk.X, pady=5)

        # X координата
        x_frame = ttk.Frame(coords_frame)
        x_frame.pack(fill=tk.X, pady=2)
        ttk.Label(x_frame, text="X:", width=3).pack(side=tk.LEFT)
        self.x_var = tk.DoubleVar(value=1.0)
        self.x_spinbox = ttk.Spinbox(x_frame, from_=-100, to=100,
                                     textvariable=self.x_var, width=10,
                                     increment=0.5)
        self.x_spinbox.pack(side=tk.LEFT, padx=5)
        self.x_slider = ttk.Scale(x_frame, from_=-10, to=10,
                                  variable=self.x_var, orient=tk.HORIZONTAL)
        self.x_slider.pack(side=tk.LEFT, fill=tk.X, expand=True)

        # Y координата
        y_frame = ttk.Frame(coords_frame)
        y_frame.pack(fill=tk.X, pady=2)
        ttk.Label(y_frame, text="Y:", width=3).pack(side=tk.LEFT)
        self.y_var = tk.DoubleVar(value=1.0)
        self.y_spinbox = ttk.Spinbox(y_frame, from_=-100, to=100,
                                     textvariable=self.y_var, width=10,
                                     increment=0.5)
        self.y_spinbox.pack(side=tk.LEFT, padx=5)
        self.y_slider = ttk.Scale(y_frame, from_=-10, to=10,
                                  variable=self.y_var, orient=tk.HORIZONTAL)
        self.y_slider.pack(side=tk.LEFT, fill=tk.X, expand=True)

        # Z координата (для 3D режиму)
        z_frame = ttk.Frame(coords_frame)
        z_frame.pack(fill=tk.X, pady=2)
        ttk.Label(z_frame, text="Z:", width=3).pack(side=tk.LEFT)
        self.z_var = tk.DoubleVar(value=0.0)
        self.z_spinbox = ttk.Spinbox(z_frame, from_=-100, to=100,
                                     textvariable=self.z_var, width=10,
                                     increment=0.5)
```

```
self.z_spinbox.pack(side=tk.LEFT, padx=5)
self.z_slider = ttk.Scale(z_frame, from_=-10, to=10,
                          variable=self.z_var, orient=tk.HORIZONTAL)
self.z_slider.pack(side=tk.LEFT, fill=tk.X, expand=True)

# Вибір кольору
color_frame = ttk.Frame(create_frame)
color_frame.pack(fill=tk.X, pady=10)
ttk.Label(color_frame, text="Колір:").pack(side=tk.LEFT)

self.color_preview = tk.Canvas(color_frame, width=30, height=20,
                               bg=self.color_palette[0],
                               highlightthickness=1)
self.color_preview.pack(side=tk.LEFT, padx=5)

ttk.Button(color_frame, text="Вибрати...",
           command=self._choose_color).pack(side=tk.LEFT)

self.current_color = self.color_palette[0]

# Опції створення
options_frame = ttk.LabelFrame(create_frame, text="Опції", padding=5)
options_frame.pack(fill=tk.X, pady=5)

self.normalize_on_create = tk.BooleanVar(value=False)
ttk.Checkbutton(options_frame, text="Нормалізувати при створенні",
                variable=self.normalize_on_create).pack(anchor=tk.W)

self.animate_on_create = tk.BooleanVar(value=True)
ttk.Checkbutton(options_frame, text="Анімувати появу",
                variable=self.animate_on_create).pack(anchor=tk.W)

# Кнопки дій
btn_frame = ttk.Frame(create_frame)
btn_frame.pack(fill=tk.X, pady=10)

ttk.Button(btn_frame, text="Створити вектор",
           command=self._create_vector,
           style="Accent.TButton").pack(fill=tk.X, pady=2)

ttk.Button(btn_frame, text="Випадковий вектор",
           command=self._create_random_vector).pack(fill=tk.X, pady=2)

ttk.Button(btn_frame, text="Одиничний вектор",
           command=self._create_unit_vector).pack(fill=tk.X, pady=2)
```

ДОДАТОК Б

Лістинг реалізація інтерактивної візуалізації векторних операцій

Лістинг Б.1 Клас системи координат

```
class CoordinateSystem:
    """Клас для управління системою координат та перетвореннями"""

    def __init__(self, canvas: tk.Canvas):
        self.canvas = canvas
        self.origin_x: float = 0
        self.origin_y: float = 0
        self.scale: float = 40.0  # пікселів на одиницю
        self.min_scale: float = 10.0
        self.max_scale: float = 200.0

        # Параметри 3D проекції
        self.rotation_x: float = 30.0  # градуси
        self.rotation_z: float = 45.0
        self.projection_distance: float = 500.0

    def update_origin(self):
        """Оновлення положення початку координат"""
        self.origin_x = self.canvas.winfo_width() / 2
        self.origin_y = self.canvas.winfo_height() / 2

    def world_to_screen_2d(self, x: float, y: float) -> Tuple[float, float]:
        """
        Перетворення світових координат у екранні (2D)

        Формула:
        screen_x = origin_x + x * scale
        screen_y = origin_y - y * scale  (Y інвертована)
        """
        screen_x = self.origin_x + x * self.scale
        screen_y = self.origin_y - y * self.scale
        return screen_x, screen_y
```

```
    def screen_to_world_2d(self, screen_x: float, screen_y: float) -> Tuple[float,
float]:
        """
        Перетворення екранних координат у світові (2D)

        Обернена формула:
        x = (screen_x - origin_x) / scale
        y = (origin_y - screen_y) / scale
        """
        x = (screen_x - self.origin_x) / self.scale
        y = (self.origin_y - screen_y) / self.scale
        return x, y

    def world_to_screen_3d(self, x: float, y: float, z: float) -> Tuple[float, float,
float]:
        """
        Перетворення 3D світових координат у екранні з урахуванням
        ізометричної проекції та обертання

        Алгоритм:
        1. Обертання навколо осі Z (rotation_z)
        2. Обертання навколо осі X (rotation_x)
        3. Ізометрична проекція на площину XY
        """
        # Конвертація кутів у радіани
        rx = math.radians(self.rotation_x)
        rz = math.radians(self.rotation_z)

        # Обертання навколо осі Z
        x1 = x * math.cos(rz) - y * math.sin(rz)
        y1 = x * math.sin(rz) + y * math.cos(rz)
        z1 = z

        # Обертання навколо осі X
        x2 = x1
        y2 = y1 * math.cos(rx) - z1 * math.sin(rx)
        z2 = y1 * math.sin(rx) + z1 * math.cos(rx)

        # Проекція на екран
        screen_x = self.origin_x + x2 * self.scale
        screen_y = self.origin_y - y2 * self.scale

        # Повертаємо також глибину для Z-сортування
        depth = z2
```

```
        return screen_x, screen_y, depth

    def apply_perspective(self, x: float, y: float, z: float) -> Tuple[float, float]:
        """
        Застосування перспективної проекції

        Формула перспективи:
        factor = distance / (distance + z')
        projected_x = x' * factor
        projected_y = y' * factor
        """
        # Спочатку застосовуємо обертання
        screen_x, screen_y, depth = self.world_to_screen_3d(x, y, z)

        # Перспективний коефіцієнт
        if self.projection_distance + depth > 0:
            factor = self.projection_distance / (self.projection_distance + depth)
        else:
            factor = 1.0

        # Застосовуємо перспективу відносно центру
        persp_x = self.origin_x + (screen_x - self.origin_x) * factor
        persp_y = self.origin_y + (screen_y - self.origin_y) * factor

        return persp_x, persp_y

    def zoom(self, factor: float, center_x: float = None, center_y: float = None):
        """
        Масштабування з центром у вказаній точці

        Алгоритм:
        1. Зберегти світові координати точки центру
        2. Змінити масштаб
        3. Обчислити нове положення точки центру
        4. Скоригувати origin для збереження точки на місці
        """
        if center_x is None:
            center_x = self.origin_x
        if center_y is None:
            center_y = self.origin_y

        # Світові координати точки фокусу до масштабування
        world_x, world_y = self.screen_to_world_2d(center_x, center_y)

        # Нове значення масштабу
```

```
        new_scale = self.scale * factor
        new_scale = max(self.min_scale, min(self.max_scale, new_scale))

        # Оновлюємо масштаб
        self.scale = new_scale

        # Обчислюємо нові екранні координати точки фокусу
        new_screen_x = self.origin_x + world_x * self.scale
        new_screen_y = self.origin_y - world_y * self.scale

        # Коригуємо origin
        self.origin_x += center_x - new_screen_x
        self.origin_y += center_y - new_screen_y
```

Лістинг Б.2 Клас відображення векторів

```
class VectorRenderer:
    """Клас для відображення векторів на Canvas"""

    # Константи відображення
    ARROW_LENGTH = 12
    ARROW_ANGLE = 25  # градуси
    LINE_WIDTH = 2
    SELECTED_WIDTH = 3
    LABEL_OFFSET = 15

    def __init__(self, canvas: tk.Canvas, coord_system: CoordinateSystem):
        self.canvas = canvas
        self.coords = coord_system
        self.rendered_items: Dict[str, List[int]] = {}  # canvas item ids

    def render_vector_2d(self, name: str, vector: 'Vector3D', color: str,
                         selected: bool = False, animated_progress: float = 1.0):
        """
        Відображення вектора у 2D режимі

        Параметри:
        - name: ім'я вектора для підпису
        - vector: об'єкт вектора
        - color: колір у форматі hex
        - selected: чи є вектор вибраним
        - animated_progress: прогрес анімації (0.0 - 1.0)
```

```
"""
# Видалення попередніх елементів
self._clear_vector(name)

items = []

# Координати з урахуванням анімації
end_x = vector.x * animated_progress
end_y = vector.y * animated_progress

# Перетворення в екранні координати
start_sx, start_sy = self.coords.world_to_screen_2d(0, 0)
end_sx, end_sy = self.coords.world_to_screen_2d(end_x, end_y)

# Параметри відображення
width = self.SELECTED_WIDTH if selected else self.LINE_WIDTH

# Малювання основної лінії вектора
line_id = self.canvas.create_line(
    start_sx, start_sy, end_sx, end_sy,
    fill=color,
    width=width,
    capstyle=tk.ROUND,
    tags=(f"vector_{name}", "vector")
)
items.append(line_id)

# Малювання наконечника стрілки
arrow_items = self._draw_arrowhead(
    start_sx, start_sy, end_sx, end_sy, color, width
)
items.extend(arrow_items)

# Малювання точки початку (якщо вибраний)
if selected:
    dot_id = self.canvas.create_oval(
        start_sx - 4, start_sy - 4,
        start_sx + 4, start_sy + 4,
        fill=color, outline="white", width=1,
        tags=(f"vector_{name}", "vector")
    )
    items.append(dot_id)

# Підпис вектора
if animated_progress >= 0.9:  # Показувати підпис після анімації
```

```
            label_items = self._draw_label(name, end_sx, end_sy, color)
            items.extend(label_items)

        self.rendered_items[name] = items

    def _draw_arrowhead(self, x1: float, y1: float, x2: float, y2: float,
                        color: str, width: float) -> List[int]:
        """
        Малювання наконечника стрілки

        Алгоритм:
        1. Обчислити кут напрямку вектора
        2. Обчислити точки наконечника під кутом ±ARROW_ANGLE
        3. Намалювати трикутник
        """
        items = []

        # Кут напрямку вектора
        dx = x2 - x1
        dy = y2 - y1
        length = math.sqrt(dx**2 + dy**2)

        if length < 1:
            return items

        angle = math.atan2(dy, dx)
        arrow_angle = math.radians(self.ARROW_ANGLE)

        # Точки наконечника
        arrow_len = self.ARROW_LENGTH

        # Ліва точка
        left_x = x2 - arrow_len * math.cos(angle - arrow_angle)
        left_y = y2 - arrow_len * math.sin(angle - arrow_angle)

        # Права точка
        right_x = x2 - arrow_len * math.cos(angle + arrow_angle)
        right_y = y2 - arrow_len * math.sin(angle + arrow_angle)

        # Малювання заповненого наконечника
        arrow_id = self.canvas.create_polygon(
            x2, y2, left_x, left_y, right_x, right_y,
            fill=color, outline=color,
            tags=("vector", "arrowhead")
        )
```

```
        items.append(arrow_id)

        return items

    def _draw_label(self, name: str, x: float, y: float, color: str) -> List[int]:
        """Малювання підпису вектора з фоном"""
        items = []

        # Зміщення підпису від кінця вектора
        offset_x = self.LABEL_OFFSET
        offset_y = -self.LABEL_OFFSET

        label_x = x + offset_x
        label_y = y + offset_y

        # Текст з обведенням для кращої видимості
        # Фон (тінь)
        shadow_id = self.canvas.create_text(
            label_x + 1, label_y + 1,
            text=f"→{name}",
            fill="#000000",
            font=("Consolas", 10, "bold"),
            anchor=tk.SW,
            tags=("vector", "label")
        )
        items.append(shadow_id)

        # Основний текст
        text_id = self.canvas.create_text(
            label_x, label_y,
            text=f"→{name}",
            fill=color,
            font=("Consolas", 10, "bold"),
            anchor=tk.SW,
            tags=("vector", "label")
        )
        items.append(text_id)

        return items

    def render_vector_3d(self, name: str, vector: 'Vector3D', color: str,
                         selected: bool = False, animated_progress: float = 1.0):
        """
        Відображення вектора у 3D режимі з проекцією
        """
```

```
            self._clear_vector(name)
            items = []

            # Координати з урахуванням анімації
            end_x = vector.x * animated_progress
            end_y = vector.y * animated_progress
            end_z = vector.z * animated_progress

            # 3D -> 2D проекція
            start_sx, start_sy, _ = self.coords.world_to_screen_3d(0, 0, 0)
            end_sx, end_sy, depth = self.coords.world_to_screen_3d(end_x, end_y,
end_z)

            # Малювання проекцій на площини (опціонально)
            if self.show_projections:
                proj_items = self._draw_projections_3d(
                    end_x, end_y, end_z, color, animated_progress
                )
                items.extend(proj_items)

            # Параметри відображення
            width = self.SELECTED_WIDTH if selected else self.LINE_WIDTH

            # Основна лінія вектора
            line_id = self.canvas.create_line(
                start_sx, start_sy, end_sx, end_sy,
                fill=color, width=width, capstyle=tk.ROUND,
                tags=(f"vector_{name}", "vector_3d")
            )
            items.append(line_id)

            # Наконечник
            arrow_items = self._draw_arrowhead(
                start_sx, start_sy, end_sx, end_sy, color, width
            )
            items.extend(arrow_items)

            # Підпис
            if animated_progress >= 0.9:
                label_items = self._draw_label(name, end_sx, end_sy, color)
                items.extend(label_items)

            self.rendered_items[name] = items

            return depth  # Повертаємо глибину для сортування
```

```
    def _draw_projections_3d(self, x: float, y: float, z: float,
                             color: str, progress: float) -> List[int]:
        """
        Малювання проекцій вектора на координатні площини
        """
        items = []
        proj_color = self._adjust_color_alpha(color, 0.3)

        # Проекція на XY площину (z=0)
        p_xy_start = self.coords.world_to_screen_3d(0, 0, 0)
        p_xy_end = self.coords.world_to_screen_3d(x * progress, y * progress, 0)

        line_xy = self.canvas.create_line(
            p_xy_start[0], p_xy_start[1],
            p_xy_end[0], p_xy_end[1],
            fill=proj_color, width=1, dash=(4, 4),
            tags=("projection",)
        )
        items.append(line_xy)

        # Вертикальна лінія від проекції до точки
        p_end = self.coords.world_to_screen_3d(x * progress, y * progress, z *
progress)
        vert_line = self.canvas.create_line(
            p_xy_end[0], p_xy_end[1],
            p_end[0], p_end[1],
            fill=proj_color, width=1, dash=(2, 2),
            tags=("projection",)
        )
        items.append(vert_line)

        return items

    def _clear_vector(self, name: str):
        """Видалення елементів вектора з Canvas"""
        if name in self.rendered_items:
            for item_id in self.rendered_items[name]:
                self.canvas.delete(item_id)
            del self.rendered_items[name]

    def clear_all(self):
        """Видалення всіх векторів"""
        for name in list(self.rendered_items.keys()):
            self._clear_vector(name)
```

Лістинг Б.3 Візуалізатор операцій

```
class OperationVisualizer:
    """Клас для візуалізації векторних операцій з анімацією"""

    def __init__(self, canvas: tk.Canvas, renderer: VectorRenderer,
                 coord_system: CoordinateSystem):
        self.canvas = canvas
        self.renderer = renderer
        self.coords = coord_system
        self.animation_items: List[int] = []

    def visualize_addition(self, v1: 'Vector3D', v2: 'Vector3D',
                           result: 'Vector3D', color1: str, color2: str,
                           animate: bool = True):
        """
        Візуалізація додавання векторів методом паралелограма

        Алгоритм:
        1. Показати перший вектор від початку координат
        2. Показати другий вектор від кінця першого (паралельне
перенесення)
        3. Анімувати появу результуючого вектора
        4. Намалювати паралелограм
        """
        self._clear_animation()

        # Екранні координати
        origin = self.coords.world_to_screen_2d(0, 0)
        end_v1 = self.coords.world_to_screen_2d(v1.x, v1.y)
        end_v2 = self.coords.world_to_screen_2d(v2.x, v2.y)
        end_result = self.coords.world_to_screen_2d(result.x, result.y)

        # Кінець v2 при паралельному перенесенні
        shifted_v2_end = self.coords.world_to_screen_2d(v1.x + v2.x, v1.y +
v2.y)

        # Малювання паралелограма (пунктирні лінії)
        # v2 від кінця v1
        para_line1 = self.canvas.create_line(
            end_v1[0], end_v1[1],
            shifted_v2_end[0], shifted_v2_end[1],
```

```
                fill=color2, width=1, dash=(5, 3),
                tags=("operation", "parallelogram")
            )
            self.animation_items.append(para_line1)

            # v1 від кінця v2
            shifted_v1_end = self.coords.world_to_screen_2d(v2.x + v1.x, v2.y +
v1.y)
            para_line2 = self.canvas.create_line(
                end_v2[0], end_v2[1],
                shifted_v1_end[0], shifted_v1_end[1],
                fill=color1, width=1, dash=(5, 3),
                tags=("operation", "parallelogram")
            )
            self.animation_items.append(para_line2)

            # Заливка паралелограма (напівпрозора)
            parallelogram = self.canvas.create_polygon(
                origin[0], origin[1],
                end_v1[0], end_v1[1],
                end_result[0], end_result[1],
                end_v2[0], end_v2[1],
                fill=self._blend_colors(color1, color2, 0.2),
                outline="",
                tags=("operation", "parallelogram_fill")
            )
            self.canvas.tag_lower(parallelogram)  # Під вектори
            self.animation_items.append(parallelogram)

            if animate:
                self._animate_result_vector(result, "#ffffff")

        def visualize_dot_product(self, v1: 'Vector3D', v2: 'Vector3D',
                                  projection: 'Vector3D', color1: str, color2: str):
            """
            Візуалізація скалярного добутку

            Показує:
            - Проекцію v1 на v2
            - Кут між векторами
            - Числове значення добутку
            """
            self._clear_animation()

            origin = self.coords.world_to_screen_2d(0, 0)
```

```
    end_v1 = self.coords.world_to_screen_2d(v1.x, v1.y)
    end_v2 = self.coords.world_to_screen_2d(v2.x, v2.y)
    end_proj = self.coords.world_to_screen_2d(projection.x, projection.y)

    # Малювання проекції
    proj_line = self.canvas.create_line(
        origin[0], origin[1],
        end_proj[0], end_proj[1],
        fill="#ffeb3b", width=3,
        tags=("operation", "projection")
    )
    self.animation_items.append(proj_line)

    # Перпендикуляр від v1 до проекції
    perp_line = self.canvas.create_line(
        end_v1[0], end_v1[1],
        end_proj[0], end_proj[1],
        fill="#9e9e9e", width=1, dash=(3, 3),
        tags=("operation", "perpendicular")
    )
    self.animation_items.append(perp_line)

    # Прямий кут (маленький квадрат)
    self._draw_right_angle(end_proj, v2, v1)

    # Дуга кута між векторами
    angle_items = self._draw_angle_arc(origin, v1, v2, color1)
    self.animation_items.extend(angle_items)

    # Текст з результатом
    dot_value = v1.dot(v2)
    angle_deg = v1.angle_between(v2)

    text_x = (origin[0] + end_v1[0] + end_v2[0]) / 3
    text_y = (origin[1] + end_v1[1] + end_v2[1]) / 3 - 20

    result_text = self.canvas.create_text(
        text_x, text_y,
        text=f"a·b = {dot_value:.2f}\n∠ = {angle_deg:.1f}°",
        fill="#ffffff", font=("Arial", 11, "bold"),
        justify=tk.CENTER,
        tags=("operation", "result_text")
    )
    self.animation_items.append(result_text)
def visualize_cross_product(self, v1: 'Vector3D', v2: 'Vector3D',
```

```
                                result: 'Vector3D', color1: str, color2: str):
        """
        Візуалізація векторного добутку

        Показує:
        - Результуючий вектор перпендикулярний до обох
        - Площу паралелограма
        - Напрямок за правилом правої руки
        """
        self._clear_animation()

        origin = self.coords.world_to_screen_3d(0, 0, 0)
        end_v1 = self.coords.world_to_screen_3d(v1.x, v1.y, v1.z)
        end_v2 = self.coords.world_to_screen_3d(v2.x, v2.y, v2.z)
        end_result = self.coords.world_to_screen_3d(result.x, result.y, result.z)

        # Паралелограм площі
        corner = self.coords.world_to_screen_3d(v1.x + v2.x, v1.y + v2.y, v1.z +
v2.z)

        area_polygon = self.canvas.create_polygon(
            origin[0], origin[1],
            end_v1[0], end_v1[1],
            corner[0], corner[1],
            end_v2[0], end_v2[1],
            fill=self._blend_colors(color1, color2, 0.3),
            outline="#ffffff", width=1,
            tags=("operation", "area")
        )
        self.canvas.tag_lower(area_polygon)
        self.animation_items.append(area_polygon)

        # Результуючий вектор (перпендикулярний)
        result_line = self.canvas.create_line(
            origin[0], origin[1],
            end_result[0], end_result[1],
            fill="#00ff00", width=3,
            arrow=tk.LAST, arrowshape=(12, 15, 5),
            tags=("operation", "cross_result")
        )
        self.animation_items.append(result_line)

        # Символ перпендикулярності
        perp_symbol = self.canvas.create_text(
            end_result[0] + 15, end_result[1] - 10,
```

```
            text="⊥",
            fill="#00ff00", font=("Arial", 14, "bold"),
            tags=("operation", "perp_symbol")
        )
        self.animation_items.append(perp_symbol)

        # Площа паралелограма
        area = result.magnitude()
        area_text = self.canvas.create_text(
            (origin[0] + corner[0]) / 2,
            (origin[1] + corner[1]) / 2,
            text=f"S = {area:.2f}",
            fill="#ffffff", font=("Arial", 10, "bold"),
            tags=("operation", "area_text")
        )
        self.animation_items.append(area_text)

    def _draw_angle_arc(self, center: Tuple[float, float],
                        v1: 'Vector3D', v2: 'Vector3D', color: str) -> List[int]:
        """Малювання дуги кута між векторами"""
        items = []

        angle1 = math.atan2(v1.y, v1.x)
        angle2 = math.atan2(v2.y, v2.x)

        # Конвертація в градуси для Canvas
        start_angle = math.degrees(angle1)
        extent = math.degrees(angle2 - angle1)

        # Нормалізація кута
        if extent > 180:
            extent -= 360
        elif extent < -180:
            extent += 360

        radius = 40

        arc = self.canvas.create_arc(
            center[0] - radius, center[1] - radius,
            center[0] + radius, center[1] + radius,
            start=-start_angle, extent=-extent,  # Інверсія для Canvas
            style=tk.ARC, outline=color, width=2,
            tags=("operation", "angle_arc")
        )
        items.append(arc)
```

```
    return items

def _animate_result_vector(self, vector: 'Vector3D', color: str,
                           duration: float = 0.5):
    """Анімація появи результуючого вектора"""
    frames = 20
    delay = int(duration * 1000 / frames)

    def animate_frame(frame: int):
        if frame > frames:
            return

        progress = frame / frames
        # Функція згладжування (ease-out)
        eased = 1 - (1 - progress) ** 3

        # Видалення попереднього кадру
        self.canvas.delete("animated_result")

        # Поточні координати
        current_x = vector.x * eased
        current_y = vector.y * eased

        origin = self.coords.world_to_screen_2d(0, 0)
        end = self.coords.world_to_screen_2d(current_x, current_y)

        # Малювання
        self.canvas.create_line(
            origin[0], origin[1],
            end[0], end[1],
            fill=color, width=3,
            arrow=tk.LAST, arrowshape=(12, 15, 5),
            tags=("animated_result",)
        )

        self.canvas.after(delay, lambda: animate_frame(frame + 1))

    animate_frame(0)

def _clear_animation(self):
    """Очищення елементів анімації"""
    for item_id in self.animation_items:
        self.canvas.delete(item_id)
    self.animation_items.clear()
```

```
        self.canvas.delete("animated_result")

    def _blend_colors(self, color1: str, color2: str, alpha: float) -> str:
        """Змішування двох кольорів з прозорістю"""
        # Парсинг hex кольорів
        r1 = int(color1[1:3], 16)
        g1 = int(color1[3:5], 16)
        b1 = int(color1[5:7], 16)

        r2 = int(color2[1:3], 16)
        g2 = int(color2[3:5], 16)
        b2 = int(color2[5:7], 16)

        # Змішування
        r = int(r1 * (1 - alpha) + r2 * alpha)
        g = int(g1 * (1 - alpha) + g2 * alpha)
        b = int(b1 * (1 - alpha) + b2 * alpha)

        return f"#{r:02x}{g:02x}{b:02x}"
```

Лістинг Б.4 Контролер анімацій

```
class AnimationController:
    """Контролер анімацій з різними функціями згладжування"""

    # Типи функцій згладжування (easing)
    EASING_FUNCTIONS = {
        'linear': lambda t: t,
        'ease_in': lambda t: t ** 2,
        'ease_out': lambda t: 1 - (1 - t) ** 2,
        'ease_in_out': lambda t: 2 * t ** 2 if t < 0.5 else 1 - (-2 * t + 2) ** 2 / 2,
        'ease_out_cubic': lambda t: 1 - (1 - t) ** 3,
        'ease_out_elastic': lambda t: (
            1 if t == 1 else
            2 ** (-10 * t) * math.sin((t * 10 - 0.75) * (2 * math.pi) / 3) + 1
        ),
        'ease_out_bounce': lambda t: AnimationController._bounce(t),
        'ease_out_back': lambda t: 1 + 2.70158 * (t - 1) ** 3 + 1.70158 * (t - 1) **
2,
    }
    def __init__(self, canvas: tk.Canvas, callback: Callable):
        self.canvas = canvas
```

```
        self.callback = callback  # Функція перемалювання
        self.animations: Dict[str, Dict] = {}
        self.is_running = False
        self.frame_rate = 60
        self.frame_delay = int(1000 / self.frame_rate)

    @staticmethod
    def _bounce(t: float) -> float:
        """Функція bounce easing"""
        n1, d1 = 7.5625, 2.75
        if t < 1 / d1:
            return n1 * t * t
        elif t < 2 / d1:
            t -= 1.5 / d1
            return n1 * t * t + 0.75
        elif t < 2.5 / d1:
            t -= 2.25 / d1
            return n1 * t * t + 0.9375
        else:
            t -= 2.625 / d1
            return n1 * t * t + 0.984375

    def start_animation(self, name: str, duration: float,
                        start_value: float, end_value: float,
                        easing: str = 'ease_out_cubic',
                        on_complete: Callable = None):
        """
        Запуск нової анімації

        Параметри:
        - name: унікальний ідентифікатор анімації
        - duration: тривалість у секундах
        - start_value, end_value: початкове та кінцеве значення
        - easing: тип функції згладжування
        - on_complete: callback після завершення
        """
        self.animations[name] = {
            'start_time': time.time(),
            'duration': duration,
            'start_value': start_value,
            'end_value': end_value,
            'current_value': start_value,
            'easing': self.EASING_FUNCTIONS.get(easing,
self.EASING_FUNCTIONS['linear']),
            'on_complete': on_complete,
```

```
            'completed': False
        }

        if not self.is_running:
            self.is_running = True
            self._animation_loop()

    def get_value(self, name: str) -> float:
        """Отримання поточного значення анімації"""
        if name in self.animations:
            return self.animations[name]['current_value']
        return 1.0  # За замовчуванням - завершена анімація

    def _animation_loop(self):
        """Головний цикл анімації"""
        current_time = time.time()
        active_animations = False

        for name, anim in list(self.animations.items()):
            if anim['completed']:
                continue

            elapsed = current_time - anim['start_time']
            progress = min(elapsed / anim['duration'], 1.0)

            # Застосування функції згладжування
            eased_progress = anim['easing'](progress)

            # Інтерполяція значення
            anim['current_value'] = (
                anim['start_value'] +
                (anim['end_value'] - anim['start_value']) * eased_progress
            )

            if progress >= 1.0:
                anim['completed'] = True
                anim['current_value'] = anim['end_value']
                if anim['on_complete']:
                    anim['on_complete']()
            else:
                active_animations = True

        # Перемалювання
        self.callback()
```

```
        # Продовження циклу
        if active_animations:
            self.canvas.after(self.frame_delay, self._animation_loop)
        else:
            self.is_running = False

    def stop_animation(self, name: str):
        """Зупинка конкретної анімації"""
        if name in self.animations:
            self.animations[name]['completed'] = True

    def stop_all(self):
        """Зупинка всіх анімацій"""
        for anim in self.animations.values():
            anim['completed'] = True
        self.is_running = False

    def animate_vector_creation(self, vector_name: str, duration: float = 0.4):
        """Анімація створення нового вектора"""
        self.start_animation(
            f"create_{vector_name}",
            duration=duration,
            start_value=0.0,
            end_value=1.0,
            easing='ease_out_back'
        )

    def animate_rotation_3d(self, target_rx: float, target_rz: float,
                            duration: float = 0.8):
        """Анімація обертання 3D сцени"""
        self.start_animation(
            "rotation_x",
            duration=duration,
            start_value=self.current_rotation_x,
            end_value=target_rx,
            easing='ease_out_cubic'
        )
        self.start_animation(
            "rotation_z",
            duration=duration,
            start_value=self.current_rotation_z,
            end_value=target_rz,
            easing='ease_out_cubic'
        )
```

ДОДАТОК В

Інтерфейс програми

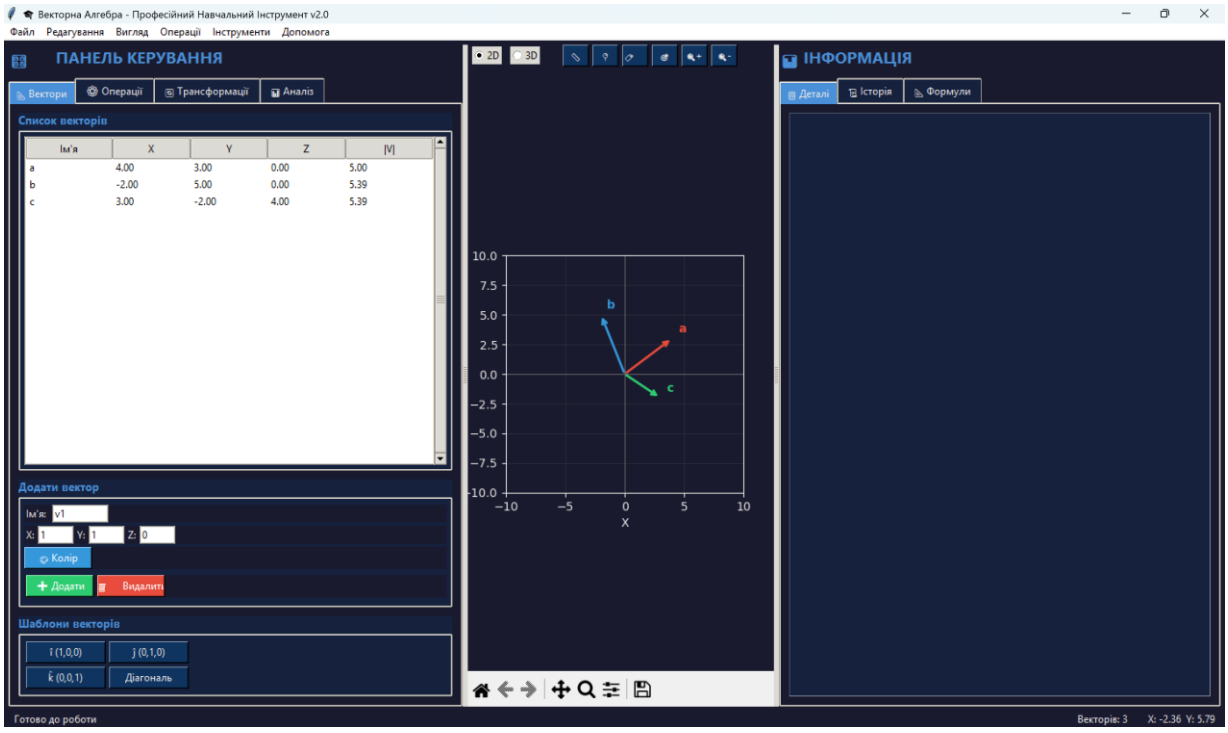


Рисунок В.1 – Головна сторінка

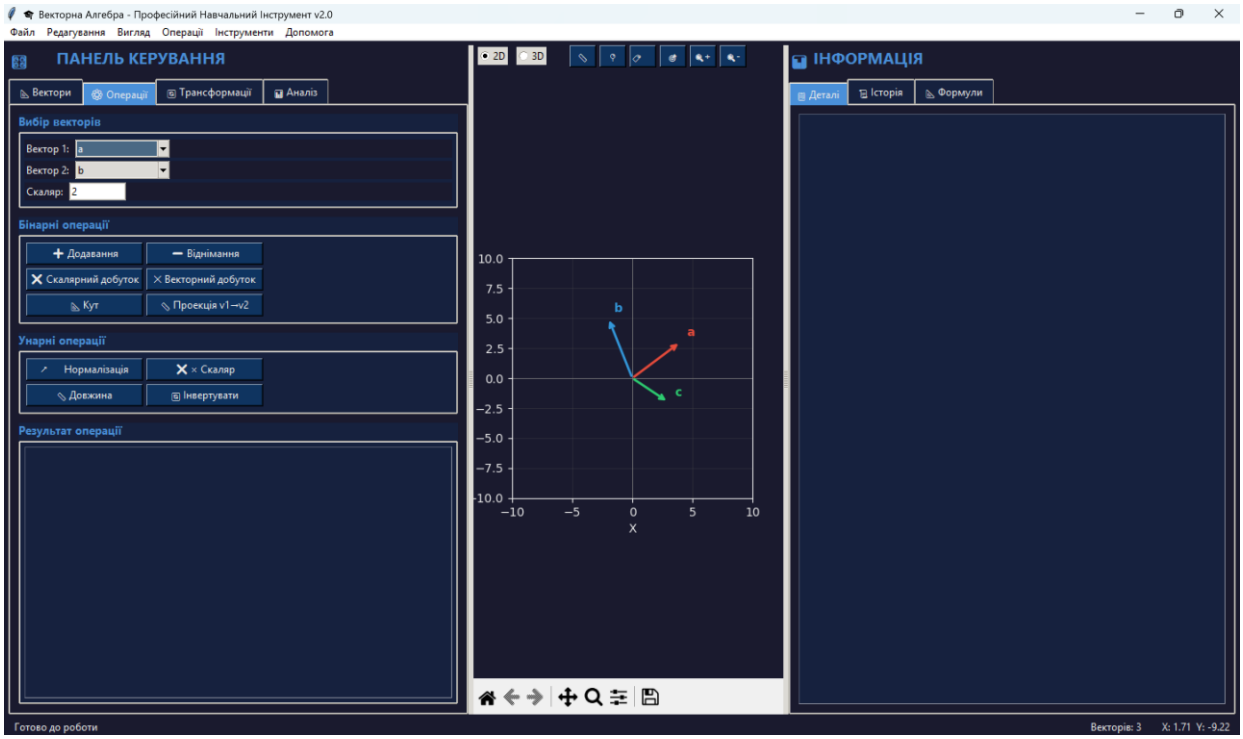


Рисунок В.2 – Сторінка операцій

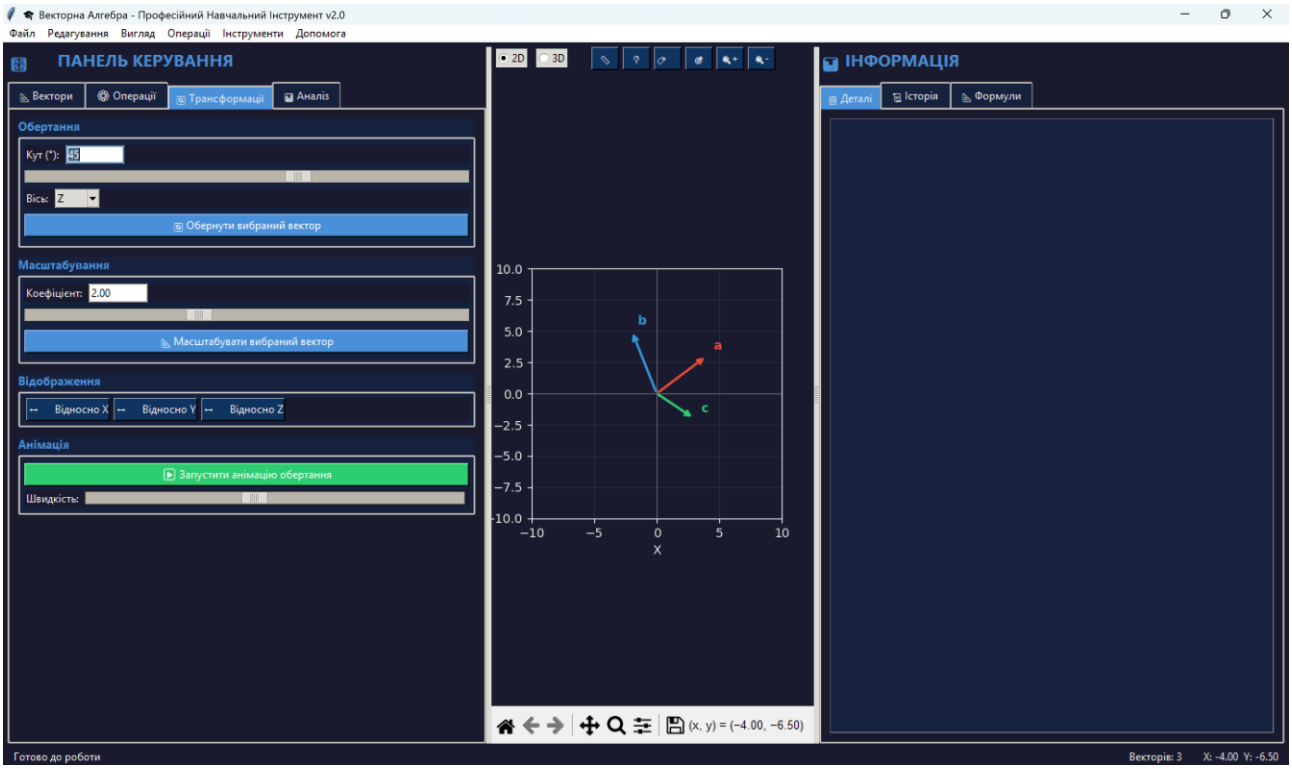


Рисунок В.3 – Сторінка трансформацій

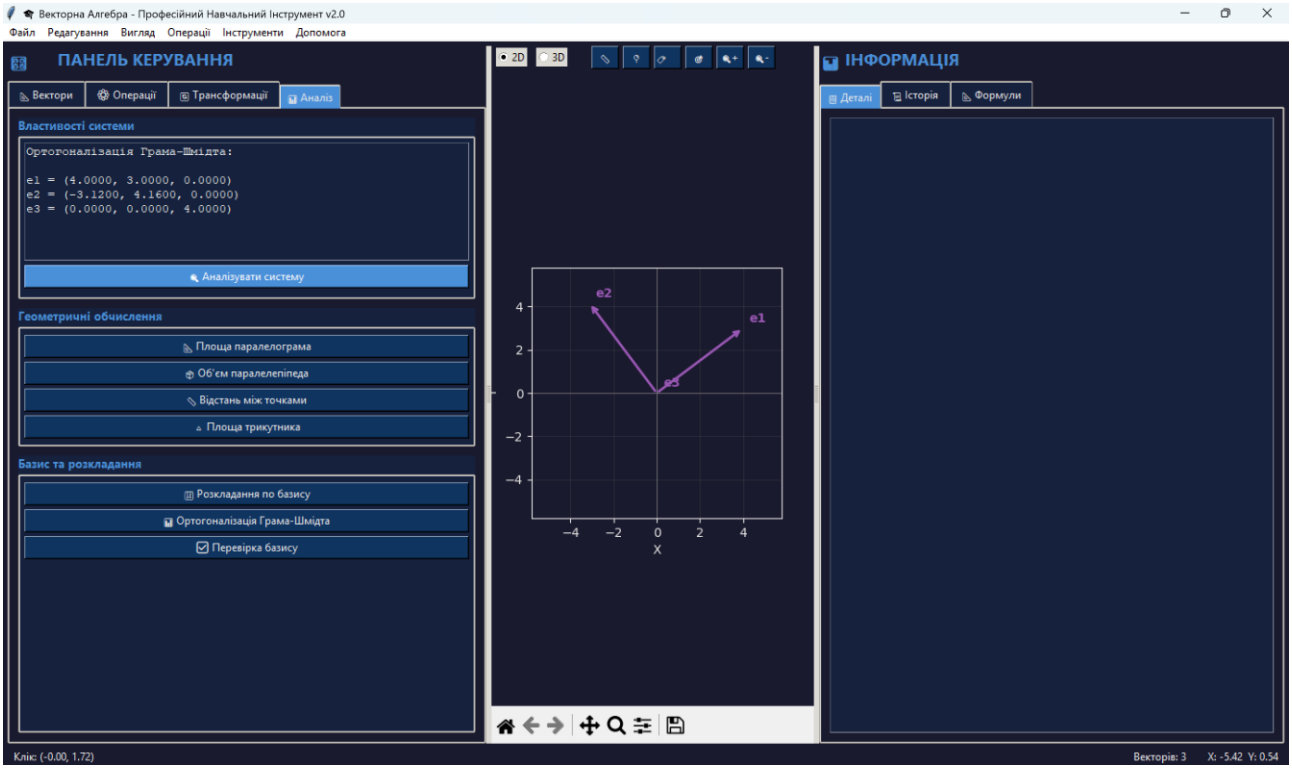


Рисунок В.4 – Сторінка аналізу

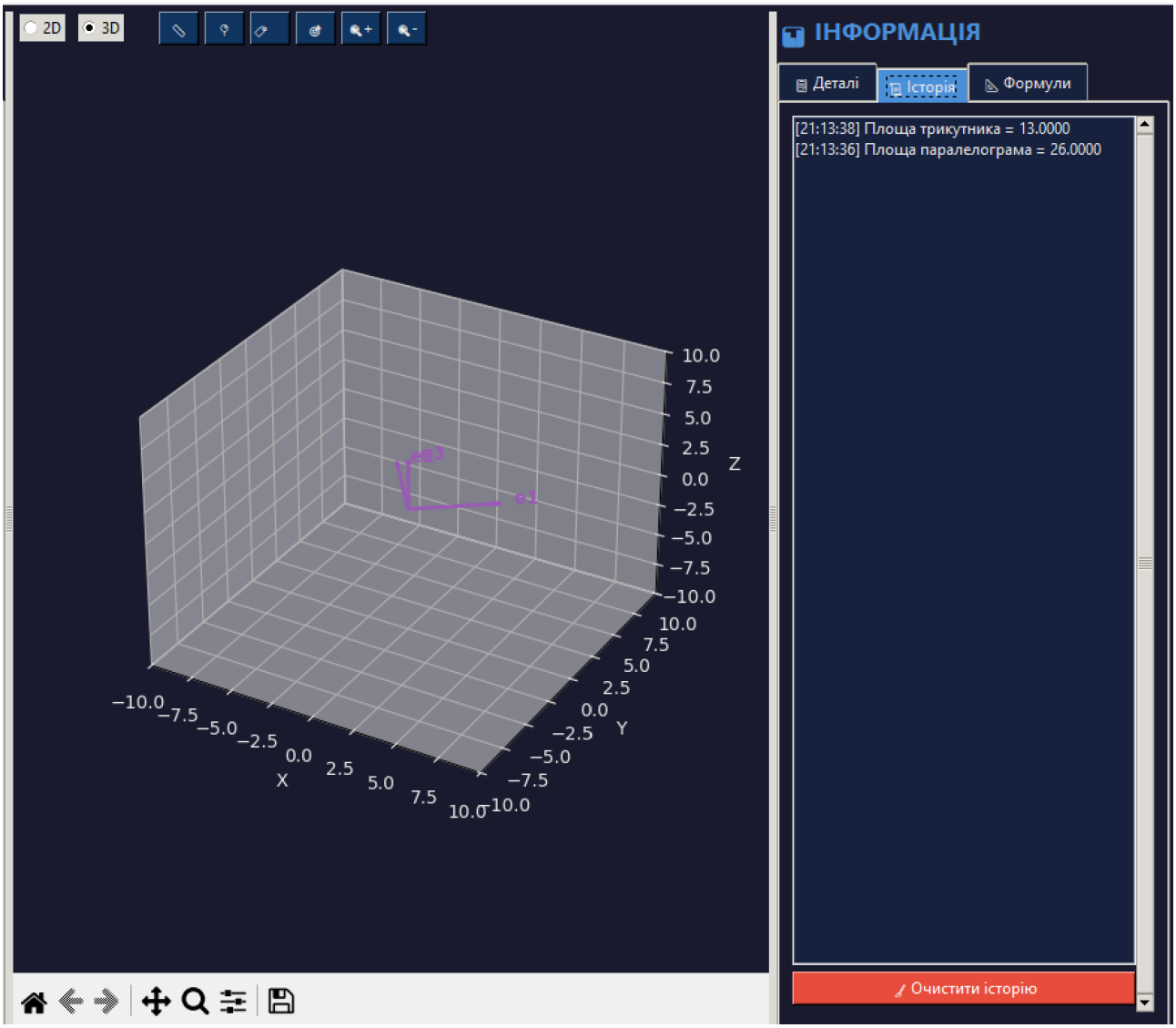


Рисунок В.5 – 3-D візуалізація та історія

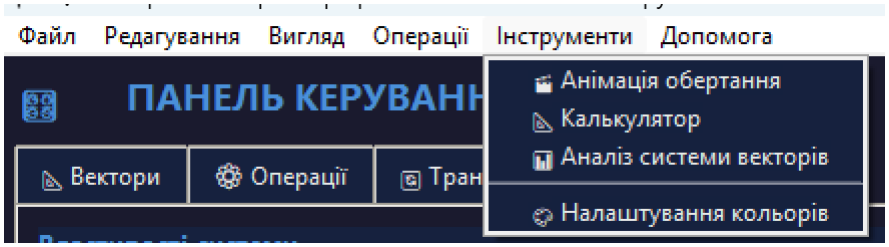


Рисунок В.6 – Панель інструментів

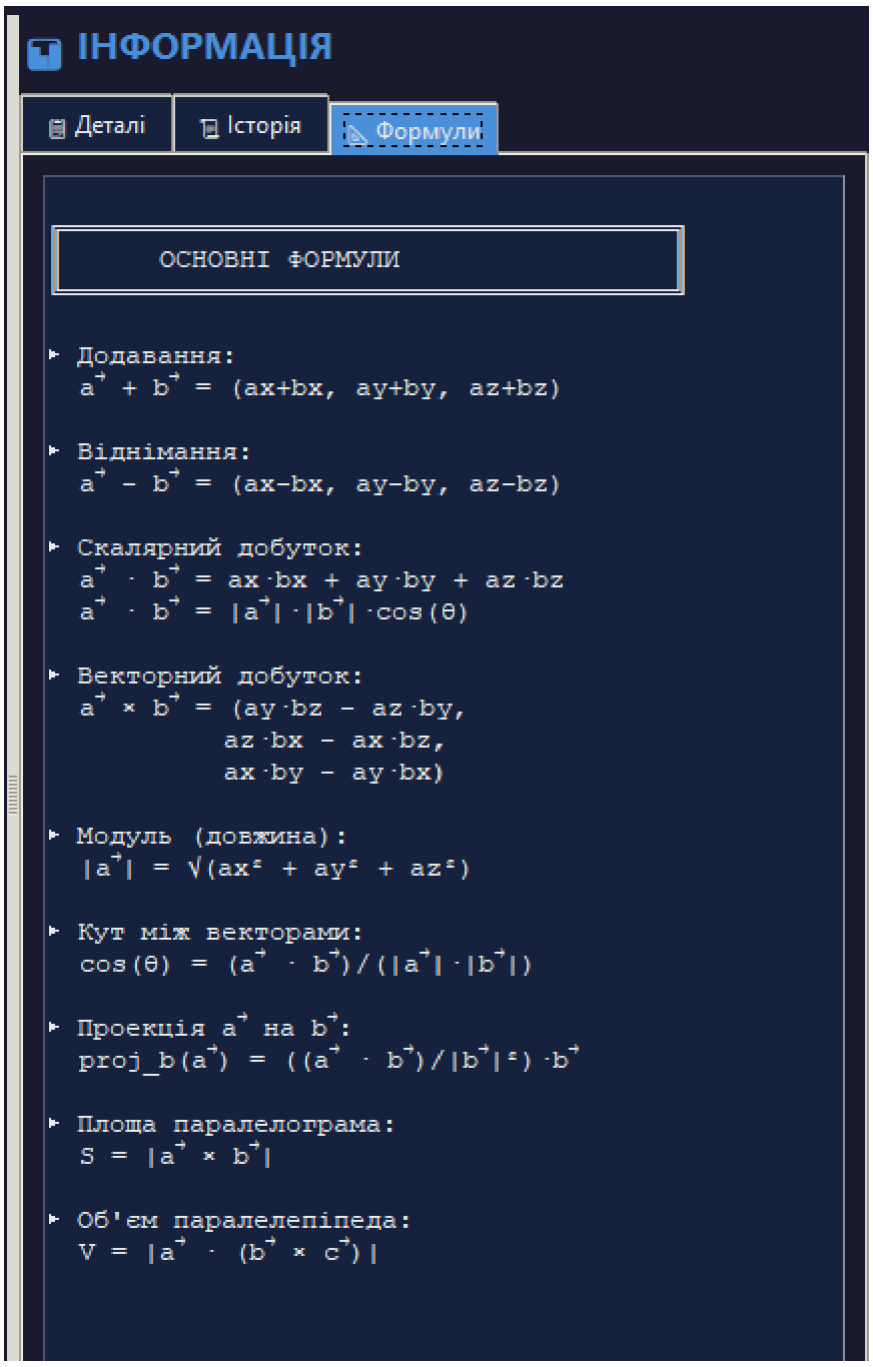


Рисунок В.7 – Сторінка формул

ДОДАТОК Г

Лістинг тестування програми

Лістинг Г.1 Тести для класу Vector3D

```
import unittest
import math
from vector_algebra_tool import Vector3D, CoordinateSystem,
VectorRenderer


class TestVector3D(unittest.TestCase):
    """Модульні тести для класу Vector3D"""

    def setUp(self):
        """Ініціалізація тестових даних"""
        self.v1 = Vector3D(3, 4, 0)
        self.v2 = Vector3D(1, 2, 3)
        self.v3 = Vector3D(0, 0, 0)
        self.v4 = Vector3D(-1, -1, -1)

    # ============ Тести базових операцій ============

    def test_magnitude_2d(self):
        """Тест обчислення довжини вектора (2D випадок)"""
        # Відомий трикутник 3-4-5
        self.assertAlmostEqual(self.v1.magnitude(), 5.0, places=10)

    def test_magnitude_3d(self):
        """Тест обчислення довжини вектора (3D)"""
        v = Vector3D(1, 2, 2)
        expected = math.sqrt(1 + 4 + 4)  # sqrt(9) = 3
        self.assertAlmostEqual(v.magnitude(), expected, places=10)

    def test_magnitude_zero_vector(self):
        """Тест довжини нульового вектора"""
        self.assertEqual(self.v3.magnitude(), 0.0)
```

```
def test_normalize(self):
    """Тест нормалізації вектора"""
    normalized = self.v1.normalize()
    self.assertAlmostEqual(normalized.magnitude(), 1.0, places=10)
    # Перевірка напрямку
    self.assertAlmostEqual(normalized.x, 3/5, places=10)
    self.assertAlmostEqual(normalized.y, 4/5, places=10)

def test_normalize_zero_vector(self):
    """Тест нормалізації нульового вектора"""
    normalized = self.v3.normalize()
    self.assertEqual(normalized.magnitude(), 0.0)

# ============ Тести арифметичних операцій ============

def test_addition(self):
    """Тест додавання векторів"""
    result = self.v1 + self.v2
    self.assertEqual(result.x, 4)
    self.assertEqual(result.y, 6)
    self.assertEqual(result.z, 3)

def test_subtraction(self):
    """Тест віднімання векторів"""
    result = self.v1 - self.v2
    self.assertEqual(result.x, 2)
    self.assertEqual(result.y, 2)
    self.assertEqual(result.z, -3)

def test_scalar_multiplication(self):
    """Тест множення на скаляр"""
    result = self.v2 * 2
    self.assertEqual(result.x, 2)
    self.assertEqual(result.y, 4)
    self.assertEqual(result.z, 6)

def test_scalar_multiplication_negative(self):
    """Тест множення на від'ємний скаляр"""
    result = self.v2 * (-1)
    self.assertEqual(result.x, -1)
    self.assertEqual(result.y, -2)
    self.assertEqual(result.z, -3)

def test_scalar_division(self):
    """Тест ділення на скаляр"""
```

```
        result = self.v2 / 2
        self.assertAlmostEqual(result.x, 0.5, places=10)
        self.assertAlmostEqual(result.y, 1.0, places=10)
        self.assertAlmostEqual(result.z, 1.5, places=10)

    def test_division_by_zero(self):
        """Тест ділення на нуль"""
        with self.assertRaises(ZeroDivisionError):
            _ = self.v1 / 0

    # ============ Тести скалярного добутку ============

    def test_dot_product(self):
        """Тест скалярного добутку"""
        v1 = Vector3D(1, 2, 3)
        v2 = Vector3D(4, 5, 6)
        # 1*4 + 2*5 + 3*6 = 4 + 10 + 18 = 32
        self.assertEqual(v1.dot(v2), 32)

    def test_dot_product_orthogonal(self):
        """Тест скалярного добутку ортогональних векторів"""
        v1 = Vector3D(1, 0, 0)
        v2 = Vector3D(0, 1, 0)
        self.assertEqual(v1.dot(v2), 0)

    def test_dot_product_parallel(self):
        """Тест скалярного добутку паралельних векторів"""
        v1 = Vector3D(2, 0, 0)
        v2 = Vector3D(3, 0, 0)
        # |v1| * |v2| * cos(0) = 2 * 3 * 1 = 6
        self.assertEqual(v1.dot(v2), 6)

    # ============ Тести векторного добутку ============

    def test_cross_product_basic(self):
        """Тест базового векторного добутку"""
        v1 = Vector3D(1, 0, 0)  # i
        v2 = Vector3D(0, 1, 0)  # j
        result = v1.cross(v2)
        # i × j = k
        self.assertEqual(result.x, 0)
        self.assertEqual(result.y, 0)
        self.assertEqual(result.z, 1)

    def test_cross_product_anticommutative(self):
```

```
    """Тест антикомутативності векторного добутку"""
    result1 = self.v1.cross(self.v2)
    result2 = self.v2.cross(self.v1)
    # a × b = -(b × a)
    self.assertAlmostEqual(result1.x, -result2.x, places=10)
    self.assertAlmostEqual(result1.y, -result2.y, places=10)
    self.assertAlmostEqual(result1.z, -result2.z, places=10)

def test_cross_product_perpendicular(self):
    """Тест перпендикулярності результату"""
    result = self.v1.cross(self.v2)
    # Результат має бути перпендикулярний до обох
    self.assertAlmostEqual(result.dot(self.v1), 0, places=10)
    self.assertAlmostEqual(result.dot(self.v2), 0, places=10)

# ============ Тести кута та проекції ============

def test_angle_between_same(self):
    """Тест кута між однаковими векторами"""
    angle = self.v1.angle_between(self.v1)
    self.assertAlmostEqual(angle, 0.0, places=5)

def test_angle_between_orthogonal(self):
    """Тест кута між ортогональними векторами"""
    v1 = Vector3D(1, 0, 0)
    v2 = Vector3D(0, 1, 0)
    angle = v1.angle_between(v2)
    self.assertAlmostEqual(angle, 90.0, places=5)

def test_angle_between_opposite(self):
    """Тест кута між протилежними векторами"""
    v1 = Vector3D(1, 0, 0)
    v2 = Vector3D(-1, 0, 0)
    angle = v1.angle_between(v2)
    self.assertAlmostEqual(angle, 180.0, places=5)

def test_projection(self):
    """Тест проекції вектора"""
    v1 = Vector3D(3, 4, 0)
    v2 = Vector3D(1, 0, 0)  # Одиничний вектор по X
    proj = v1.project_onto(v2)
    # Проекція (3,4,0) на X = (3,0,0)
    self.assertAlmostEqual(proj.x, 3.0, places=10)
    self.assertAlmostEqual(proj.y, 0.0, places=10)
    self.assertAlmostEqual(proj.z, 0.0, places=10)
```

```
    # ============ Тести властивостей ============

    def test_is_zero(self):
        """Тест перевірки нульового вектора"""
        self.assertTrue(self.v3.is_zero())
        self.assertFalse(self.v1.is_zero())

    def test_is_collinear(self):
        """Тест перевірки колінеарності"""
        v1 = Vector3D(1, 2, 3)
        v2 = Vector3D(2, 4, 6)  # Колінеарний (v1 * 2)
        v3 = Vector3D(1, 2, 4)  # Не колінеарний
        self.assertTrue(v1.is_collinear(v2))
        self.assertFalse(v1.is_collinear(v3))

    def test_is_orthogonal(self):
        """Тест перевірки ортогональності"""
        v1 = Vector3D(1, 0, 0)
        v2 = Vector3D(0, 1, 0)
        v3 = Vector3D(1, 1, 0)
        self.assertTrue(v1.is_orthogonal(v2))
        self.assertFalse(v1.is_orthogonal(v3))


class TestCoordinateSystem(unittest.TestCase):
    """Тести для системи координат"""

    def setUp(self):
        """Ініціалізація"""
        # Створюємо mock canvas
        import tkinter as tk
        self.root = tk.Tk()
        self.root.withdraw()
        self.canvas = tk.Canvas(self.root, width=800, height=600)
        self.coord_system = CoordinateSystem(self.canvas)
        self.coord_system.origin_x = 400
        self.coord_system.origin_y = 300
        self.coord_system.scale = 40

    def tearDown(self):
        self.root.destroy()

    def test_world_to_screen_origin(self):
        """Тест перетворення початку координат"""
```

```
            sx, sy = self.coord_system.world_to_screen_2d(0, 0)
            self.assertEqual(sx, 400)
            self.assertEqual(sy, 300)

        def test_world_to_screen_positive(self):
            """Тест перетворення додатних координат"""
            sx, sy = self.coord_system.world_to_screen_2d(1, 1)
            # x: 400 + 1*40 = 440
            # y: 300 - 1*40 = 260 (інвертована)
            self.assertEqual(sx, 440)
            self.assertEqual(sy, 260)

        def test_screen_to_world_roundtrip(self):
            """Тест зворотного перетворення"""
            original_x, original_y = 2.5, -1.5
            sx, sy = self.coord_system.world_to_screen_2d(original_x, original_y)
            wx, wy = self.coord_system.screen_to_world_2d(sx, sy)
            self.assertAlmostEqual(wx, original_x, places=10)
            self.assertAlmostEqual(wy, original_y, places=10)

        def test_zoom_in(self):
            """Тест збільшення масштабу"""
            old_scale = self.coord_system.scale
            self.coord_system.zoom(1.5)
            self.assertEqual(self.coord_system.scale, old_scale * 1.5)

        def test_zoom_limits(self):
            """Тест обмежень масштабування"""
            # Спроба надмірного збільшення
            self.coord_system.zoom(100)
            self.assertLessEqual(self.coord_system.scale,
self.coord_system.max_scale)

            # Спроба надмірного зменшення
            self.coord_system.zoom(0.001)
            self.assertGreaterEqual(self.coord_system.scale,
self.coord_system.min_scale)


    class TestVectorOperations(unittest.TestCase):
        """Інтеграційні тести для операцій над векторами"""

        def test_parallelogram_rule(self):
            """Тест правила паралелограма для додавання"""
            v1 = Vector3D(2, 1, 0)
```

```
        v2 = Vector3D(1, 3, 0)

        # Сума
        result = v1 + v2

        # Перевірка діагоналі паралелограма
        self.assertEqual(result.x, 3)
        self.assertEqual(result.y, 4)

    def test_gram_schmidt(self):
        """Тест ортогоналізації Грама-Шмідта"""
        v1 = Vector3D(1, 1, 0)
        v2 = Vector3D(1, 0, 0)

        # Ортогоналізація
        u1 = v1.normalize()
        u2_proj = v2.project_onto(u1)
        u2 = (v2 - u2_proj).normalize()

        # Перевірка ортогональності
        self.assertAlmostEqual(u1.dot(u2), 0, places=10)

        # Перевірка нормалізації
        self.assertAlmostEqual(u1.magnitude(), 1.0, places=10)
        self.assertAlmostEqual(u2.magnitude(), 1.0, places=10)

    def test_triple_scalar_product(self):
        """Тест мішаного добутку (об'єм паралелепіпеда)"""
        v1 = Vector3D(1, 0, 0)
        v2 = Vector3D(0, 1, 0)
        v3 = Vector3D(0, 0, 1)

        # Мішаний добуток: (v1 × v2) · v3
        cross = v1.cross(v2)
        volume = abs(cross.dot(v3))

        # Об'єм одиничного куба = 1
        self.assertAlmostEqual(volume, 1.0, places=10)


if __name__ == '__main__':
    # Запуск тестів з детальним виводом
    unittest.main(verbosity=2)
```

Лістинг Г.2 Інтеграційні тести

```
class TestIntegration(unittest.TestCase):
    """Інтеграційні тести всієї системи"""

    @classmethod
    def setUpClass(cls):
        """Створення тестового екземпляра застосунку"""
        cls.root = tk.Tk()
        cls.root.withdraw()  # Приховати вікно
        cls.app = VectorAlgebraApp.__new__(VectorAlgebraApp)
        cls.app.__init__()

    @classmethod
    def tearDownClass(cls):
        cls.app.destroy()

    def test_create_and_render_vector(self):
        """Тест створення вектора та його відображення"""
        # Створення вектора
        self.app._create_vector_internal("test_v", 3, 4, 0, "#ff0000")

        # Перевірка наявності
        self.assertIn("test_v", self.app.vectors)

        # Перевірка значень
        v = self.app.vectors["test_v"]
        self.assertEqual(v.x, 3)
        self.assertEqual(v.y, 4)
        self.assertEqual(v.z, 0)

        # Перевірка рендерингу
        self.assertIn("test_v", self.app.renderer.rendered_items)

    def test_vector_selection(self):
        """Тест вибору вектора"""
        self.app._create_vector_internal("sel_test", 2, 2, 0, "#00ff00")
        self.app._select_vector("sel_test")

        self.assertEqual(self.app.selected_vector, "sel_test")

    def test_operation_chain(self):
        """Тест ланцюжка операцій"""
        # Створення двох векторів
```

```
        self.app._create_vector_internal("a", 1, 0, 0, "#ff0000")
        self.app._create_vector_internal("b", 0, 1, 0, "#00ff00")

        # Додавання
        self.app._perform_addition("a", "b", "sum_ab")

        # Перевірка результату
        self.assertIn("sum_ab", self.app.vectors)
        result = self.app.vectors["sum_ab"]
        self.assertEqual(result.x, 1)
        self.assertEqual(result.y, 1)

    def test_undo_redo(self):
        """Тест системи Undo/Redo"""
        initial_count = len(self.app.vectors)

        # Створення вектора
        self.app._create_vector_internal("undo_test", 5, 5, 0, "#0000ff")
        self.assertEqual(len(self.app.vectors), initial_count + 1)

        # Undo
        self.app._undo()
        self.assertEqual(len(self.app.vectors), initial_count)

        # Redo
        self.app._redo()
        self.assertEqual(len(self.app.vectors), initial_count + 1)

    def test_mode_switching(self):
        """Тест перемикання 2D/3D режиму"""
        # Початковий режим - 2D
        self.assertFalse(self.app.is_3d_mode)

        # Перемикання на 3D
        self.app._set_mode(True)
        self.assertTrue(self.app.is_3d_mode)

        # Назад на 2D
        self.app._set_mode(False)
        self.assertFalse(self.app.is_3d_mode)

    def test_save_load_project(self):
        """Тест збереження та завантаження проекту"""
        import tempfile
        import os
```

```
        # Створення тестових даних
        self.app._clear_all()
        self.app._create_vector_internal("save_test", 3, 4, 5, "#ff00ff")

        # Збереження
        with tempfile.NamedTemporaryFile(mode='w', suffix='.json',
                                         delete=False) as f:
            temp_path = f.name

        try:
            self.app._save_to_file(temp_path)

            # Очищення
            self.app._clear_all()
            self.assertEqual(len(self.app.vectors), 0)

            # Завантаження
            self.app._load_from_file(temp_path)

            # Перевірка
            self.assertIn("save_test", self.app.vectors)
            v = self.app.vectors["save_test"]
            self.assertEqual(v.x, 3)
            self.assertEqual(v.y, 4)
            self.assertEqual(v.z, 5)
        finally:
            os.unlink(temp_path)

    def test_canvas_interaction(self):
        """Тест взаємодії з Canvas"""
        # Симуляція кліку
        event = type('Event', (), {
            'x': 400,
            'y': 300,
            'state': 0
        })()

        # Клік у центрі (початок координат)
        self.app._on_canvas_click(event)

        # Перевірка оновлення координат у статусному рядку
        # (координати повинні бути близько до 0,0)

    def test_zoom_interaction(self):
```

```
"""Тест масштабування"""
initial_scale = self.app.scale

# Симуляція прокрутки колеса миші
event = type('Event', (), {
    'x': 400,
    'y': 300,
    'delta': 120,  # Windows
    'num': 4       # Linux
})()

self.app._on_mouse_wheel(event)

# Масштаб має збільшитися
self.assertGreater(self.app.scale, initial_scale)
```